Extrapolation-based Path Invariants for Abstraction Refinement of Fifo Systems

Alexander Heussner, Grégoire Sutre, Tristan Le Gall

Laboratoire Bordelais de Recherche en Informatique
UMR CNRS 5800 – Université de Bordeaux
351 cours de la Libération
33405 Talence cedex, France

Research Report RR-1459-09

Extrapolation-based Path Invariants for Abstraction Refinement of Fifo Systems

Alexander Heußner[1,2], Tristan Le Gall[3]

Grégoire Sutre[1]

[1] LaBRI, CNRS, Université Bordeaux 1
[2] Unviersité Libre de Bruxelles
[3] CEA LIST, DILS/LMeASI

{heussner,sutre}@labri.fr

tristan.le-gall@cea.fr

Mai 2009

This report is the long and extended version of the paper with the same title that later appeared in the Proceedings of the 16th International SPIN Workshop on Model Checking of Software, Grenoble June 2009, published in LNCS 5578, pp.107–124, Springer, 2009.

Revised and extended version of the report: June 2011.

Extrapolation-based Path Invariants for Abstraction Refinement of Fifo Systems*

Alexander Heußner[1,2], Tristan Le Gall[3], and Grégoire Sutre[1]

[1] LaBRI, Université Bordeaux, Cnrs
[2] Université Libre de Bruxelles
[3] CEA LIST, DILS/LMeASI
{heussner, sutre}@labri.fr tristan.le-gall@cea.fr

Abstract. The technique of counterexample-guided abstraction refinement (Cegar) has been successfully applied in the areas of software and hardware verification. Automatic abstraction refinement is also desirable for the safety verification of complex infinite-state models. This techreport investigates Cegar in the context of formal models of network protocols, in our case, the verification of fifo systems. Our main contribution is the introduction of *extrapolation-based path invariants* for abstraction refinement. We develop a range of algorithms that are based on this novel theoretical notion, and which are parameterized by different extrapolation operators. These are utilized as subroutines in the refinement step of our Cegar semi-algorithm that is based on recognizable partition abstractions. We give sufficient conditions for the termination of Cegar by constraining the extrapolation operator. Our empirical evaluation confirms the benefit of extrapolation-based path invariants.

1 Introduction

Distributed processes that communicate over a network of reliable and unbounded fifo channels are an important model for the automatic verification of client-server architectures and network protocols. As easy as this model seems at a first glance, as hard is the verification of communication protocols in general: distributed processes that run in parallel and that exchange messages in an asynchronous way, therefore exhibiting complex interactions, allow for a gargantuan (and sometimes infinite) number of possible—emergent—behaviors. Hence, verifying these multitude of behaviors is far beyond any checking by hand and, regarding the emergence of behaviors, not directly deducible from the originally given set of simple processes; consequently, verification is not possible without automatic methods and the support of algorithmic tools.

Fifo Systems

We focus on communicating fifo systems that consist of a set of finite automata that model the processes, and a set of reliable, unbounded fifo queues that model

* This work was partly supported by ANR project AVERISS (ANR-06-SETIN-001).

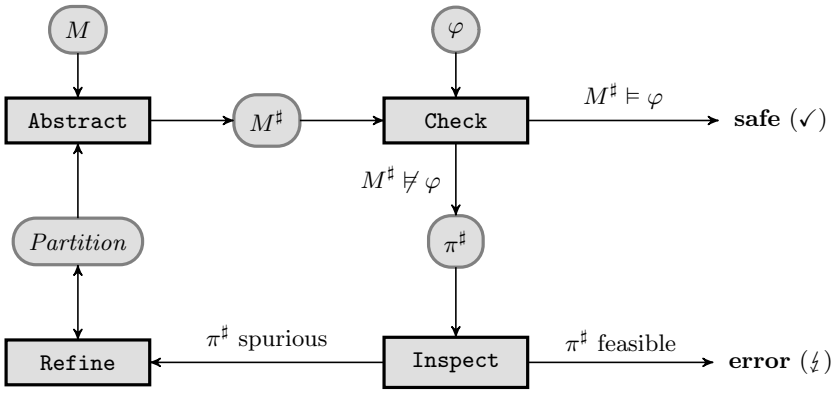


Fig. 1: Basic Steps of the CEGAR Loop

the communication channels. This class of infinite-state systems is, unfortunately, Turing-complete already in the case of one fifo queue [BZ83]. In general, two approaches for the automatic verification of Turing-complete infinite-state models have been considered in the literature: (a) exact semi-algorithms that compute forward or backward reachability sets (e.g., [BG99, BH99, FIS03] for fifo systems) but may not terminate, and (b) algorithms that always terminate but only compute an over-approximation of these reachability sets (e.g., [LGJJ06, YBCI08] for fifo systems).

CEGAR

In the last decade, counterexample-guided abstraction refinement (CEGAR) has emerged as a powerful technique that bridges the gap between these two approaches [CGJ+03]. CEGAR plays a prominent role in the automatic, iterative approximation and refinement of abstractions and has been applied successfully in the areas of software [BR01, HJMS02] and hardware verification [CGJ+03]. Briefly, the CEGAR approach to the verification of a safety property φ for an (infinite-state) model M, i.e., the decision whether $M \models \varphi$, consists in an abstract–check–refine loop that iterates the four following steps:

1. build a safety conservative, finite-state abstraction $M^\sharp$ of the model (e.g., a predicate abstraction [GS97] which partitions the state space);
2. model-check the abstraction against the given safety property (check whether $M^\sharp \models \varphi$): if the abstraction is safe, then so is the original model (return ✓), otherwise a finite counterexample path $\pi^\sharp$ is found in the abstraction;
3. symbolically simulate the abstract counterexample on the original model: if the counterexample is *feasible* then the original model is unsafe (return ↯), otherwise it is *spurious* (i.e., a false negative) and
4. extract a refinement for $M^\sharp$ that rules out the spurious counterexample before iterating this procedure (jump back to 1).

The crucial part in CEGAR-based verification is refinement, which must find a new partition that is both (1) precise enough to rule out the spurious counterexample and (2) computationally "simple". In most techniques, refinement is based on the generation of *path invariants*; these are invariants along the spurious counterexample that prove its unfeasibility.

Our Contribution

We present several generic algorithms to obtain path invariants based on parameterized extrapolation operators. A path invariant is given by a series of recognizable languages represented by QDDs [BG99], generated by an "extrapolation" of the actual spurious counterexample. Our path invariant generation procedures are fully generic with respect to the extrapolation; in our experiments, we implemented those algorithms with two different extrapolation operators adapted to fifo systems.

We formally present the resulting CEGAR semi-algorithm, which uses the path-invariants in order to refine the current partition. This operation consists in splitting abstract states that occur on the counterexample with the generated path invariant. We also give partial termination results that, in contrast to the classical CEGAR literature, do not rely on an "a priori finiteness condition" on the set of all possible abstractions. Actually, our results depend mainly on our generic extrapolation-based path invariant generation. In particular we show that our semi-algorithm always terminates if (at least) one of these two conditions is satisfied: (1) the fifo system under verification is unsafe, or (2) it has a finite reachability set and the parameterized extrapolation has a finite image for each value of the parameter. We cannot expect termination in general since safety verification is known to be undecidable for fifo systems [BZ83].

We have implemented our approach as part of the McScM framework [McScM]. Our tool performs CEGAR-based safety verification of fifo systems. Experimental results on a suite of (small to medium size) network protocols allow for a first discussion of our approach's advantages.

Related Work

Exact semi-algorithms for reachability set computations of fifo systems usually apply *acceleration techniques* [BG99, BH99, FIS03] that, intuitively, compute the effect of iterating a given "control flow" loop. The tools LASH [Lash] (for counter/fifo systems) and TREX [TReX] (for lossy fifo systems) implement these techniques. However, recognizable languages equipped with Presburger formulas (CQDDs [BH99]) are required to represent (and compute) the effect of *counting* loops [BG99, FIS03]. Moreover such tools may only terminate when the fifo system can be flattened into an equivalent system without nested loops. Our experiments show that our approach can cope with both counting loops and nested loops that cannot be flattened.

A totally different approach is presented in [VSVA04a] which combines a learning based approach for over-approximating the set of reachable control

states with safety verification. Even if this approach is shown to be only complete when a given abstraction of the fifo system's trace language is regular, it nevertheless allows to derive counter-examples in a semi-algorithmic way. A prototypical implementation can be found as part of [Lever].

The closest approach to ours is *abstract regular model checking* (ARMC), an extension of the generic regular model-checking framework based on the abstract–check–refine paradigm [BHV04]. As in classical regular model-checking, a system is modeled as follows: configurations are words over a finite alphabet and the transition relation is given by a finite-state transducer. The analysis consists in an over-approximated forward exploration (by Kleene iteration), followed, in case of a non-empty intersection with the bad states, by an exact backward computation along the reached sets. Two parametrized automata abstraction schemes are provided in [BHV04], both based on state merging. These schemes fit in our definition of extrapolation, and therefore can also be used in our framework. Notice that in ARMC, abstraction is performed on the data structures that are used to represent sets of configurations, whereas in our case the system itself is abstracted. After each refinement step, ARMC restarts (from scratch) the approximated forward exploration from the refined reached set, whereas our refinement is *local* to the spurious counterexample path. Moreover, the precision of the abstraction is *global* in ARMC, and may only increase (for the entire system) at each refinement step. In contrast, our path invariant generation procedures only use the precision *required* for each spurious counterexample. First benchmarks demonstrate the benefit of our local and adaptive approach for the larger examples, where a "highly" precise abstraction is required only for a few control loops. Last, our approach is not tied to words and automata. In this work we only focus on fifo systems, but our framework is fully generic and could be applied to other infinite-state systems (e.g., hybrid systems), provided that suitable parameterized extrapolations are designed (e.g., on polyhedra).

Outline

We recapitulate fifo systems in Section 2 and define their partition abstractions in Section 3. Refinement and extrapolation-based generation of path invariants are developed in Section 4. The latter also provides an overview of the extrapolation used in our implementation. In Sections 5 and 6, we present the general CEGAR semi-algorithm, and analyze its correctness and termination. Experimental results are presented in Section 7, along with some perspectives.

For the sake of completeness, all results are proved in detail as necessary, additional material can be found in the appendix. This technical report is the detailed and enhanced version of the results presented at the SPIN workshop on model checking software 2009 [HLS09].

Acknowledgments. We thank the anonymous reviewers of the primary publication for supporting and guiding the genesis of this publication, and we are

especially grateful for the fruitful and challenging discussions with Jérôme Leroux and Anca Muscholl.

2 Fifo Systems

This section presents basic definitions and notations for fifo systems that will be used throughout the paper.

For any set S we write $\wp(S)$ for the set of all subsets of S, and S^n for the set of n-tuples over S (when $n \geq 1$). For any $i \in \{1, \ldots, n\}$, we denote by $\boldsymbol{s}(i)$ the i^{th} *component* of an n-tuple $\boldsymbol{s}$. Given $\boldsymbol{s} \in S^n$, $i \in \{1, \ldots, n\}$ and $u \in S$, we write $\boldsymbol{s}[i \leftarrow u]$ for the n-tuple $\boldsymbol{s'} \in S^n$ defined by $\boldsymbol{s'}(i) = u$ and $\boldsymbol{s'}(j) = \boldsymbol{s}(j)$ for all $j \in \{1, \ldots, n\}$ with $j \neq i$.

Let Σ denote an *alphabet* (i.e., a non-empty set of *letters*). We write Σ^* for the set of all *finite words* (*words* for short) over Σ, and we let ε denote the *empty word*. For any two words $w, w' \in \Sigma^*$, we write $w \cdot w'$ for their *concatenation*. A *language* is any subset of Σ^*. For any language L, we denote by L^* its *Kleene closure* and we write $L^+ = L \cdot L^*$. The *alphabet of* L, written $alph(L)$, is the least subset A of Σ such that $L \subseteq A^*$. For any word $w \in \Sigma^*$, the singleton language $\{w\}$ will be written simply as word w when no confusion is possible.

2.1 Safety Verification of Labeled Transition Systems

We will use labeled transition systems to formally define the behavioral semantics of fifo systems. A *labeled transition system* is any triple $LTS = \langle \mathcal{C}, \Sigma, \rightarrow \rangle$ where $\mathcal{C}$ is a set of *configurations*, Σ is a finite set of *actions* and $\rightarrow \subseteq \mathcal{C} \times \Sigma \times \mathcal{C}$ is a (labeled) *transition relation*. We say that LTS is *finite* when $\mathcal{C}$ is finite. For simplicity, we will often write $c \xrightarrow{l} c'$ in place of $(c, l, c') \in \rightarrow$.

A *finite path* (*path* for short) in LTS is any pair $\pi = (c, u)$ where $c \in \mathcal{C}$, and u is either the empty sequence, or a non-empty finite sequence of transitions $(c_0, l_0, c'_0), \ldots, (c_{h-1}, l_{h-1}, c'_{h-1})$ such that $c_0 = c$ and $c'_{i-1} = c_i$ for every $0 < i < h$. We simply write π as $c_0 \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} c_h$. The natural number h is called the *length* of π. We say that π is a *simple path* if $c_i \neq c_j$ for all $0 \leq i < j \leq h$. For any two sets $Init \subseteq \mathcal{C}$ and $Bad \subseteq \mathcal{C}$ of configurations, a *path from* $Init$ *to* Bad is any path $c_0 \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} c_h$ such that $c_0 \in Init$ and $c_h \in Bad$. Observe that if $c \in Init \cap Bad$ then c is a path (of zero length) from $Init$ to Bad. The *reachability set* of LTS from $Init$ is the set of configurations c such that there is a path from $Init$ to $\{c\}$.

In this paper, we focus on the verification of safety properties on fifo systems. A safety property is in general specified as a set of "bad" configurations that should not be reachable from the initial configurations. Formally, a *safety condition* for a labeled transition system $LTS = \langle \mathcal{C}, \Sigma, \rightarrow \rangle$ is a pair $(Init, Bad)$ of subsets of $\mathcal{C}$. We say that LTS is $(Init, Bad)$*-unsafe* if there is a path from $Init$ to Bad in LTS, which is called a *counterexample*. We say that LTS is $(Init, Bad)$*-safe* when it is not $(Init, Bad)$-unsafe.

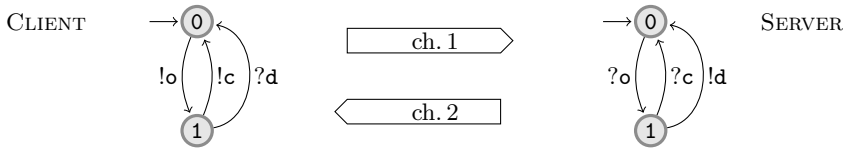


Fig. 2: The Connection/Disconnection Protocol [JR86]

2.2 Fifo Systems

The asynchronous communication of distributed systems is usually modeled as a set of local processes together with a network topology given by channels between processes. Each process can be modeled by a finite-state machine that sends and receives messages on the channels to which it is connected. Let us consider a classical example, which will be used in the remainder of this paper to illustrate our approach.

Example 2.1. The connection/disconnection protocol [JR86]—in the following abbreviated as c/d protocol—between two hosts is depicted in Figure 2. This model is composed of two processes, a client and a server, as well as two unidirectional channels. The client can open a session by sending the message `open` (abbreviated `o`) to the server and changing its state to `1` (session established). Afterwards, he may close it either actively by sending the message `c(lose)`, or passively as a reaction to the `d(isconnect)` message from the server. The server receives the request to establish a shared session by the message `open` and thereupon enters its state `1` (session on server-side established). He as well can either actively or passively close the session by sending a `disconnect` request or by receiving a `close`. ◇

To simplify the presentation, we restrict our attention to the case of one finite-state control process. The general case of multiple processes can be reduced to this simpler form by taking the asynchronous product of all processes. For the connection/disconnection protocol, the asynchronous product of the two processes is depicted in Figure 3. For instance, the global control state `11` combines the local "session established" control state of both peers.

We assume that channels respect the fifo semantics for send and receive actions, and, hence, we call them "queues" in the remainder of the paper.

Definition 2.2. *A* fifo system $\mathcal{A}$ *is a* 4*-tuple* $\langle Q, M, n, \Delta \rangle$ *where:*

- Q *is a finite set of* control states*,*
- M *is a finite alphabet of* messages*,*
- $n \geq 1$ *is the number of fifo queues,*
- $\Delta \subseteq Q \times \Sigma \times Q$ *is a set of* transition rules*,*
  where $\Sigma = \{1, \ldots, n\} \times \{!, ?\} \times M$ *is the set of* fifo actions *over* n *queues.*

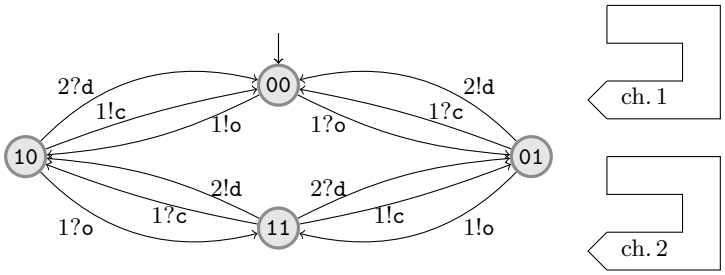


Fig. 3: Fifo System Representing the Connection/Disconnection Protocol

Simplifying notation, fifo actions in Σ will be shortly written $i!m$ and $i?m$ instead of $(i, !, m)$ and $(i, ?, m)$. The intended meaning of fifo actions is the following: $i!m$ represents the "sending of message m on queue i" and $i?m$ denotes the "reception of message m from queue i". The operational semantics of a fifo system $\mathcal{A}$ is formally given by its associated labeled transition system $[\![\mathcal{A}]\!]$ defined below.

Definition 2.3. *The* operational semantics *of a fifo system $\mathcal{A} = \langle Q, M, n, \Delta\rangle$ is the labeled transition system $[\![\mathcal{A}]\!] = \langle \mathcal{C}, \Sigma, \rightarrow\rangle$ defined as follows:*

- $\mathcal{C} = Q \times (M^*)^n$ *is the set of configurations,*
- $\Sigma = \{1, \ldots, n\} \times \{!, ?\} \times M$ *is the set of actions,*
- *the transition relation $\rightarrow \subseteq \mathcal{C} \times \Sigma \times \mathcal{C}$ is the set of triples $((q, \boldsymbol{w}), l, (q', \boldsymbol{w'}))$ such that $(q, l, q') \in \Delta$ and that satisfy the two following conditions:*
  - *if $l = i!m$ then $\boldsymbol{w'}(i) = \boldsymbol{w}(i) \cdot m$ and $\boldsymbol{w'}(j) = \boldsymbol{w}(j)$ for all $j \neq i$,*
  - *if $l = i?m$ then $\boldsymbol{w}(i) = m \cdot \boldsymbol{w'}(i)$ and $\boldsymbol{w'}(j) = \boldsymbol{w}(j)$ for all $j \neq i$.*

The configurations of $\mathcal{C}$ can be seen as momentary snapshots of the whole system: each configuration includes the current control state and the current queue contents. The transition relation between configurations captures the effect of send and receive actions on queues, ensuring the fifo ordering of actions: messages sent to a queue are received in the same order; further, a receive action can only be taken if the appropriate message is at front of the queue.

Example 2.4. The fifo system $\mathcal{A} = \langle\{\mathtt{00}, \mathtt{01}, \mathtt{10}, \mathtt{11}\}, \{\mathtt{o}, \mathtt{c}, \mathtt{d}\}, 2, \Delta\rangle$ that corresponds to the c/d protocol is displayed in Figure 3. Its operational semantics is presented in Figure 4. The set of initial configurations is $Init = \{(\mathtt{00}, \varepsilon, \varepsilon)\}$. A set of bad configurations for this protocol is $Bad = \{\mathtt{00}, \mathtt{10}\} \times (\mathtt{c} \cdot M^* \times M^*)$. This set contains configurations where the server is in local state `0` but the first message in the first queue is `close`. This is the classical case of an *undefined reception* which results in a *(local) deadlock* for the server. Setting the initial configuration to $c_0 = (\mathtt{00}, \varepsilon, \varepsilon)$, a counterexample to the safety condition $(\{c_0\}, Bad)$ is the path $(\mathtt{00}, \varepsilon, \varepsilon) \xrightarrow{1!\mathtt{o}} (\mathtt{10}, \mathtt{o}, \varepsilon) \xrightarrow{1?\mathtt{o}} (\mathtt{11}, \varepsilon, \varepsilon) \xrightarrow{2!\mathtt{d}} (\mathtt{10}, \varepsilon, \mathtt{d}) \xrightarrow{1!\mathtt{c}} (\mathtt{00}, \mathtt{c}, \mathtt{d})$ in $[\![\mathcal{A}]\!]$. As can be deduced from Figure 4, no counterexample has less than four transitions. Further, there is an infinite path that never visits the same configuration twice nor reaches Bad, e.g., by alternating the actions 1!o and 1!c. ◇

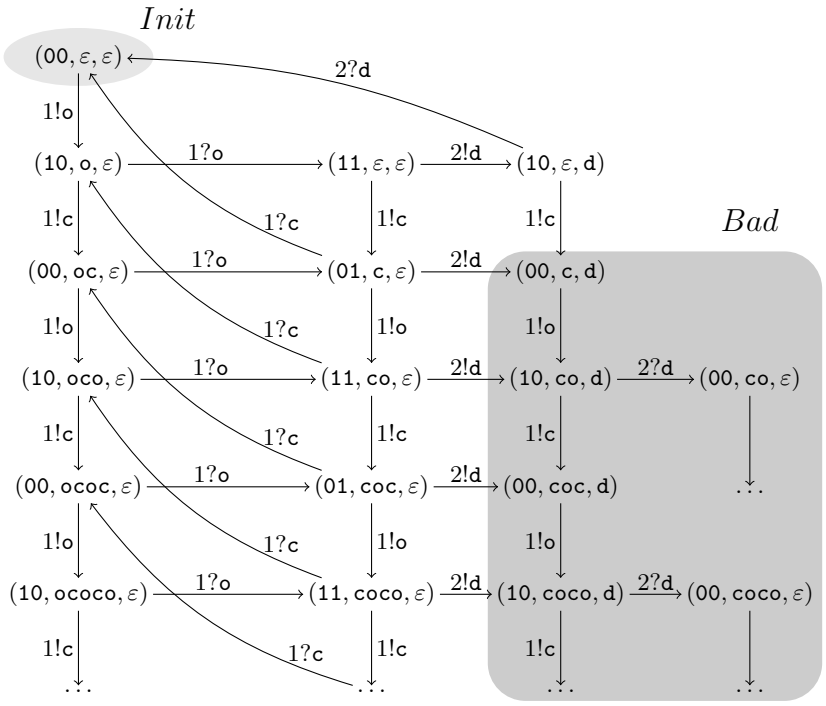


Fig. 4: Operational Semantics of the C/D Protocol [JR86] (Example 2.4)

3 Partition Abstraction for Fifo Systems

In the context of CEGAR-based safety verification, automatic abstraction techniques are usually based on predicates [GS97] or partitions [CGJ+03]. In this work, we focus on partition-based abstraction and refinement techniques for fifo systems. Still, our extrapolation-based path invariant generation techniques could also be used in the context of predicate-based abstractions.

A *partition* of a set S is any set P of non-empty pairwise disjoint subsets of S such that $S = \bigcup_{p \in P} p$. Elements p of a partition P are called *classes*. For any element s in S, we denote by $[\, s \,]_P$ the class in P containing s.

At the labeled transition system level, partition abstraction consists of merging configurations that are equivalent with respect to a given equivalence relation, or a given partition. In practice, it is often desirable to maintain different partitions for different control states, to keep partition sizes relatively small (as in predicate abstraction of programs, where predicates are local to each control point). We follow this approach in our definition of partition abstraction for fifo systems, by associating a partition of $(M^*)^n$ with each control state. To ease notation, we write $\overline{L} = (M^*)^n \setminus L$ for the *complement* of any subset L of $(M^*)^n$.

To effectively compute partition abstractions for fifo systems, we need a family of finitely representable subsets of $(M^*)^n$. A natural candidate is the class

of recognizable subsets of $(M^*)^n$, or, equivalently, of QDD-definable subsets of $(M^*)^n$ [BGWW97], since this class is effectively closed under Boolean operations. (The definition and main properties of QDDs are recalled in Appendix A.) Recall Mezei's theorem that states that a subset L of $(M^*)^n$ is *recognizable* if (and only if) it is a finite union of subsets of the form $L_1 \times \cdots \times L_n$ where each L_i is a regular language over M [Ber79]. We extend recognizability in the natural way to subsets of the set $\mathcal{C} = Q \times (M^*)^n$ of configurations. A subset $C \subseteq \mathcal{C}$ is *recognizable* if $\{\boldsymbol{w} \,|\, (q, \boldsymbol{w}) \in C\}$ is recognizable for every $q \in Q$. We denote by $\mathcal{R}ec\,((M^*)^n)$ the set of recognizable subsets of $(M^*)^n$, and write $\mathbb{P}\,((M^*)^n)$ for the set of all finite partitions of $(M^*)^n$ where classes are recognizable subsets of $(M^*)^n$.

Definition 3.1. *Consider a fifo system $\mathcal{A} = \langle Q, M, n, \Delta\rangle$ and a* partition map *$P : Q \to \mathbb{P}\,((M^*)^n)$. The* partition abstraction *of $[\![\mathcal{A}]\!]$ induced by P is the finite labeled transition system $[\![\mathcal{A}]\!]^\sharp_P = \langle \mathcal{C}^\sharp_P, \Sigma, \to^\sharp_P\rangle$ defined as follows:*

- *$\mathcal{C}^\sharp_P = \{(q,p) \mid q \in Q$ and $p \in P(q)\}$ is the set of abstract configurations,*
- *$\Sigma = \{1,\ldots,n\} \times \{!,?\} \times M$ is the set of actions,*
- *the abstract transition relation $\to^\sharp_P \subseteq \mathcal{C}^\sharp_P \times \Sigma \times \mathcal{C}^\sharp_P$ is the set of triples $((q,p), l, (q',p'))$ such that $(q,\boldsymbol{w}) \xrightarrow{l} (q',\boldsymbol{w'})$ for some $\boldsymbol{w} \in p$ and $\boldsymbol{w'} \in p'$.*

To relate concrete and abstract configurations, we define the *abstraction function* $\alpha_P : \mathcal{C} \to \mathcal{C}^\sharp_P$, and its extension to $\wp(\mathcal{C}) \to \wp(\mathcal{C}^\sharp_P)$, as well as the *concretization function* $\gamma_P : \mathcal{C}^\sharp_P \to \mathcal{C}$, extended to $\wp(\mathcal{C}^\sharp_P) \to \wp(\mathcal{C})$, as expected:

$$\begin{array}{rclcrcl} \alpha_P((q,\boldsymbol{w})) & = & (q, [\,\boldsymbol{w}\,]_{P(q)}) & \qquad & \alpha_P(C) & = & \{\alpha(c) \mid c \in C\} \\ \gamma_P((q,p)) & = & \{q\} \times p & & \gamma_P(C^\sharp) & = & \bigcup \{\gamma(c^\sharp) \;\big|\; c^\sharp \in C^\sharp\} \end{array}$$

To simplify notations, we shall drop the P subscript when the partition map can easily be derived from the context. Intuitively, an abstract configuration (q,p) of $[\![\mathcal{A}]\!]^\sharp$ represents the set $\{q\} \times p$ of (concrete) configurations of $[\![\mathcal{A}]\!]$. The abstract transition relation $\to^\sharp$ is the existential lift of the concrete transition relation $\to$ to abstract configurations.

The following forward and backward language transformers will be used to capture the effect of fifo actions. The functions $post : \Sigma \times \wp((M^*)^n) \to \wp((M^*)^n)$ and $pre : \Sigma \times \wp((M^*)^n) \to \wp((M^*)^n)$ are defined by:

$$\begin{array}{rcl} post(i!m, L) & = & \{\boldsymbol{w}[i \leftarrow u] \mid \boldsymbol{w} \in L, u \in M^* \text{ and } \boldsymbol{w}(i) \cdot m = u\} \\ post(i?m, L) & = & \{\boldsymbol{w}[i \leftarrow u] \mid \boldsymbol{w} \in L, u \in M^* \text{ and } \boldsymbol{w}(i) = m \cdot u\} \\ pre(i!m, L) & = & \{\boldsymbol{w}[i \leftarrow u] \mid \boldsymbol{w} \in L, u \in M^* \text{ and } \boldsymbol{w}(i) = u \cdot m\} \\ pre(i?m, L) & = & \{\boldsymbol{w}[i \leftarrow u] \mid \boldsymbol{w} \in L, u \in M^* \text{ and } m \cdot \boldsymbol{w}(i) = u\} \end{array}$$

Obviously, $post(l, L)$ and $pre(l, L)$ are effectively recognizable subsets of $(M^*)^n$ for any $l \in \Sigma$ and any recognizable subset $L \subseteq (M^*)^n$. Moreover, we may use *post* and *pre* to characterize the abstract transition relation of a partition abstraction $[\![\mathcal{A}]\!]^\sharp_P$, as follows: for any rule $(q, l, q') \in \Delta$ and for any pair $(p, p') \in P(q) \times P(q')$, we have $(q,p) \xrightarrow{l}{}^\sharp (q',p')$ iff $post(l,p) \cap p' \neq \emptyset$ iff $p \cap pre(l,p') \neq \emptyset$.

Lemma 3.2. *For any fifo system $\mathcal{A}$ and partition map $P : Q \to \mathbb{P}((M^*)^n)$, $[\![\mathcal{A}]\!]^\sharp$ is effectively computable. For any recognizable subset $C \subseteq \mathcal{C}$, $\alpha(C)$ is effectively computable.*

Proof. The lemma follows from (1) closure under intersection, complement and *post* (or *pre*) of recognizable subsets of $(M^*)^n$, and (2) decidability of emptiness for recognizable subsets of $(M^*)^n$. □

We extend α to paths in the obvious way: $\alpha(c_0 \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} c_h) = \alpha(c_0) \xrightarrow{l_0}^\sharp \cdots \xrightarrow{l_{h-1}}^\sharp \alpha(c_h)$. Observe that $\alpha(\pi)$ is an abstract path in $[\![\mathcal{A}]\!]^\sharp$ for any concrete path π in $[\![\mathcal{A}]\!]$. We therefore obtain the following safety preservation property.

Proposition 3.3. *Consider a fifo system $\mathcal{A}$ and a safety condition $(Init, Bad)$ for $[\![\mathcal{A}]\!]$. For any partition abstraction $[\![\mathcal{A}]\!]^\sharp$ of $[\![\mathcal{A}]\!]$, if $[\![\mathcal{A}]\!]^\sharp$ is $(\alpha(Init), \alpha(Bad))$-safe then $[\![\mathcal{A}]\!]$ is $(Init, Bad)$-safe.*

Proof. If $[\![\mathcal{A}]\!]$ is $(Init, Bad)$-unsafe then there is a path π in $[\![\mathcal{A}]\!]$ from $Init$ to Bad, and hence $\alpha(\pi)$ is an abstract path from $\alpha(Init)$ to $\alpha(Bad)$ in $[\![\mathcal{A}]\!]^\sharp$. □

The converse to this proposition does not hold in general. An abstract counterexample $\pi^\sharp$ is called *feasible* if there exists a concrete counterexample π such that $\pi^\sharp = \alpha(\pi)$, and $\pi^\sharp$ is called *spurious* otherwise.

Lemma 3.4. *For any fifo system $\mathcal{A}$, any partition map $P : Q \to \mathbb{P}((M^*)^n)$, and any safety condition $(Init, Bad)$ for $[\![\mathcal{A}]\!]$, feasibility of abstract counterexamples is effectively decidable.*

Proof. Given an abstract counterexample $\pi^\sharp = (q_0, p_0) \xrightarrow{l_0}^\sharp \cdots \xrightarrow{l_{h-1}}^\sharp (q_h, p_h)$, we deduce from the definition of feasibility that $\pi^\sharp$ is feasible iff the subset $L \subseteq (M^*)^n$ defined below is non-empty:

$$L \;=\; p_h \cap post(l_{h-1}, (p_{h-1} \cap \cdots \cap post(l_1, p_1 \cap post(l_0, p_0 \cap Init)) \cdots)) \cap Bad$$

Since L is an effectively computable recognizable subset of $(M^*)^n$, we may effectively decide whether L is non-empty, which concludes the proof. □

Example 3.5. Continuing the discussion of the c/d protocol, we consider the partition abstraction induced by the following partition map:

$q \in Q$	$\mathtt{00}$	$\mathtt{10}$	$\mathtt{01}$	$\mathtt{11}$
$P(q)$	$\varepsilon \times \varepsilon,\ \overline{\varepsilon \times \varepsilon}$	$\mathtt{o}^* \times \varepsilon,\ \overline{\mathtt{o}^* \times \varepsilon}$	$M^* \times M^*$	$M^* \times M^*$

The set of initial abstract configurations is $\alpha(Init) = \{(\mathtt{00}, \varepsilon \times \varepsilon)\}$, and the set of bad abstract configurations is $\alpha(Bad) = \{(\mathtt{00}, \overline{\varepsilon \times \varepsilon}), (\mathtt{10}, \overline{\mathtt{o}^* \times \varepsilon})\}$. The resulting partition abstraction is the finite labeled transition system depicted in Figure 5. A simple graph search reveals several abstract counterexamples, for instance $\pi^\sharp = (\mathtt{00}, \varepsilon \times \varepsilon) \xrightarrow{1!\mathtt{o}}^\sharp (\mathtt{10}, \mathtt{o}^* \times \varepsilon) \xrightarrow{1!\mathtt{c}}^\sharp (\mathtt{00}, \overline{\varepsilon \times \varepsilon})$. This counterexample is spurious since the only concrete path that corresponds to $\pi^\sharp$ (i.e., whose image under α is $\pi^\sharp$) is $\pi = (\mathtt{00}, \varepsilon, \varepsilon) \xrightarrow{1!\mathtt{o}} (\mathtt{10}, \mathtt{o}, \varepsilon) \xrightarrow{1!\mathtt{c}} (\mathtt{00}, \mathtt{oc}, \varepsilon) \notin Bad$. ◇

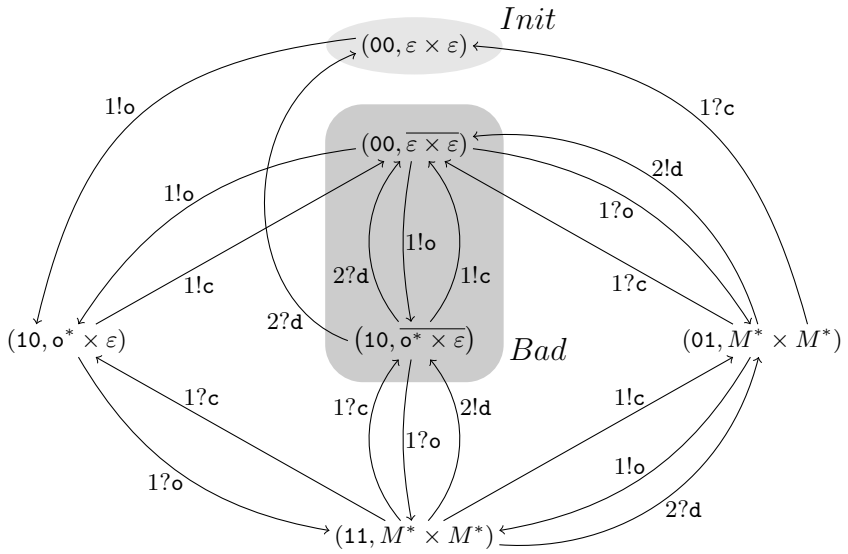


Fig. 5: Example Partition Abstraction of the C/D Protocol (Example 3.5)

4 Counterexample-based Generation of Path Invariants

The abstraction-based verification of safety properties relies on refinement techniques that gradually increase the precision of abstractions in order to rule out spurious abstract counterexamples. Refinement for partition abstractions simply consists in splitting some classes into a sub-partition.

4.1 Partition Refinement

Given two partitions P and $\widetilde{P}$ of a set S, we say that $\widetilde{P}$ *refines* P when each class $\widetilde{p} \in \widetilde{P}$ is contained in some class $p \in P$. Moreover we then write $[\widetilde{p}]_P$ for the class $p \in P$ containing $\widetilde{p}$.

Let us fix, for the remainder of this section, a fifo system $\mathcal{A} = \langle Q, M, n, \Delta \rangle$ and a safety condition $(Init, Bad)$ for $[\![\mathcal{A}]\!]$. Given two partition maps $P, \widetilde{P} : Q \to \mathbb{P}((M^*)^n)$, we say that $\widetilde{P}$ *refines* P if $\widetilde{P}(q)$ refines $P(q)$ for every control state $q \in Q$. If $\widetilde{P}$ refines P, then for any abstract path $(q_0, \widetilde{p}_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, \widetilde{p}_h)$ in $[\![\mathcal{A}]\!]^\sharp_{\widetilde{P}}$, it holds that $(q_0, [\widetilde{p}_0]_{P(q_0)}) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, [\widetilde{p}_h]_{P(q_h)})$ is an abstract path in $[\![\mathcal{A}]\!]^\sharp_P$. This fact shows that, informally, refining a partition abstraction does not introduce any new spurious counterexample.

When a spurious counterexample is found in the abstraction, the partition map must be refined so as to rule out this counterexample. We formalize this concept for an abstract path $\pi^\sharp = (q_0, p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, p_h)$ in $[\![\mathcal{A}]\!]^\sharp_P$ from $\alpha_P(Init)$ to $\alpha_P(Bad)$ as follows: a refinement $\widetilde{P}$ of P is said to *rule out* the

abstract counterexample $\pi^\sharp$ if there exists no path $\widetilde{\pi}^\sharp = (q_0, \widetilde{p}_0) \xrightarrow{l_0}^\sharp \cdots \xrightarrow{l_{h-1}}^\sharp (q_h, \widetilde{p}_h)$ from $\alpha_{\widetilde{P}}(Init)$ to $\alpha_{\widetilde{P}}(Bad)$ in $[\![\mathcal{A}]\!]^\sharp_{\widetilde{P}}$ satisfying $\widetilde{p}_i \subseteq p_i$ for all $0 \leq i \leq h$. Note that if $\pi^\sharp$ is a feasible counterexample, then no refinement of P can rule it out. Conversely, if $\widetilde{P}$ is a refinement of P that rules out $\pi^\sharp$ then any refinement of $\widetilde{P}$ also rules out $\pi^\sharp$.

The main challenge in CEGAR is the discovery of "suitable" refinements, that are computationally "simple" but "precise enough" to rule out spurious counterexamples. In this work, we focus on counterexample-guided refinements based on path invariants.

Definition 4.1. *Consider a partition map P and a spurious counterexample $\pi^\sharp = (q_0, p_0) \xrightarrow{l_0}^\sharp \cdots \xrightarrow{l_{h-1}}^\sharp (q_h, p_h)$ in $[\![\mathcal{A}]\!]^\sharp_P$. A* path invariant *for $\pi^\sharp$ is any sequence $L_0, \ldots, L_h$ of recognizable subsets of $(M^*)^n$ such that:*

(i) we have $(\{q_0\} \times p_0) \cap Init \subseteq \{q_0\} \times L_0$, and
(ii) we have $p_{i+1} \cap post(l_i, p_i \cap L_i) \subseteq L_{i+1}$ for every $0 \leq i < h$, and
(iii) we have $(\{q_h\} \times (p_h \cap L_h)) \cap Bad = \emptyset$

Observe that condition *(ii)* is more general than $post(l_i, L_i) \subseteq L_{i+1}$ which is classically required for inductive invariants. With this relaxed condition, path invariants are tailored to the given spurious counterexample, and therefore can be simpler (e.g., be coarser or have more empty L_i).

Proposition 4.2. *Consider a partition map P and a simple spurious counterexample $\pi^\sharp = (q_0, p_0) \xrightarrow{l_0}^\sharp \cdots \xrightarrow{l_{h-1}}^\sharp (q_h, p_h)$. Given a path invariant $L_0, \ldots, L_h$ for $\pi^\sharp$, the partition map $\widetilde{P}$ defined below is a refinement of P that rules out $\pi^\sharp$:*

$$\widetilde{P}(q) \;=\; (P(q) \setminus \{p_i \mid i \in I(q)\}) \;\cup \bigcup_{i \in I(q)} \left\{p_i \cap L_i, p_i \cap \overline{L_i}\right\} \setminus \{\emptyset\}$$

where $I(q) = \{i \mid 0 \leq i \leq h, q_i = q\}$ for each control state $q \in Q$.

Proof. For any control sate $q \in Q$, since $\pi^\sharp$ is simple, we have $p_i = p_j \Rightarrow i = j$ for every $i, j \in I(q)$. The function $\widetilde{P}$ defined in the proposition is therefore a partition map that refines P by definition. We need to show that $\widetilde{P}$ rules out $\pi^\sharp$. By contradiction, assume there exists a path $\widetilde{\pi}^\sharp = (q_0, \widetilde{p}_0) \xrightarrow{l_0}^\sharp \cdots \xrightarrow{l_{h-1}}^\sharp (q_h, \widetilde{p}_h)$ from $\alpha_{\widetilde{P}}(Init)$ to $\alpha_{\widetilde{P}}(Bad)$ in $[\![\mathcal{A}]\!]^\sharp_{\widetilde{P}}$ satisfying $\widetilde{p}_i \subseteq p_i$ for all $0 \leq i \leq h$.

We first show that $\widetilde{p}_i \in \left\{p_i \cap L_i, p_i \cap \overline{L_i}\right\}$ for every $0 \leq i \leq h$. Consider any integer i with $0 \leq i \leq h$. Observe that $i \in I(q_i)$. If $\widetilde{p}_i \in P(q_i)$ then $\widetilde{p}_i = p_i$ as $\widetilde{p}_i \subseteq p_i$. Hence, $\widetilde{p}_i \notin (P(q_i) \setminus \{p_j \mid j \in I(q_i)\})$. Since $\widetilde{p}_i \in \widetilde{P}(q_i)$, we obtain that $\widetilde{p}_i \in \left\{p_j \cap L_j, p_j \cap \overline{L_j}\right\}$ for some $j \in I(q_i)$. Let us now prove that $i = j$. Remark that $q_i = q_j$ as $j \in I(q_i)$. Moreover, we get $\widetilde{p}_i \subseteq p_j$, and hence $\widetilde{p}_i \subseteq p_i \cap p_j$. Therefore $p_i = p_j$ since p_i and p_j are classes of the same partition $P(q_i)$. We arrive at $(q_i, p_i) = (q_j, p_j)$ which implies that $i = j$ since $\pi^\sharp$ is simple. We have thus shown that $\widetilde{p}_i \in \left\{p_i \cap L_i, p_i \cap \overline{L_i}\right\}$ for every $0 \leq i \leq h$.

Recall that $L_0, \ldots, L_h$ is a path invariant for $\pi^\sharp$. We prove by induction on i that $\widetilde{p}_i = p_i \cap L_i$ for every $0 \leq i \leq h$. For the basis, we derive from item (i) of Definition 4.1 that $\{q_0\} \times (p_0 \cap \overline{L_0})$ is disjoint from $Init$. Since $(q_0, \widetilde{p}_0) \in \alpha_{\widetilde{P}}(Init)$, we get that $\{q_0\} \times \widetilde{p}_0$ intersects $Init$. Therefore $\widetilde{p}_0 \neq p_0 \cap \overline{L_0}$, and hence $\widetilde{p}_0 = p_0 \cap L_0$. For the induction step, assume that $\widetilde{p}_i = p_i \cap L_i$ for some $0 \leq i < h$. We have $p_{i+1} \cap post(l_i, \widetilde{p}_i) \subseteq L_{i+1}$ according to item (ii) of Definition 4.1. Therefore, we get that $p_{i+1} \cap \overline{L_{i+1}}$ is disjoint from $post(l_i, \widetilde{p}_i)$. Since $(q_i, \widetilde{p}_i) \xrightarrow{l_i}{}^\sharp (q_{i+1}, \widetilde{p}_{i+1})$ is an abstract transition in $[\![\mathcal{A}]\!]^\sharp_{\widetilde{P}}$, we get that $\widetilde{p}_{i+1}$ intersects $post(l_i, \widetilde{p}_i)$. Therefore $\widetilde{p}_{i+1} \neq p_{i+1} \cap \overline{L_{i+1}}$, and hence $\widetilde{p}_{i+1} = p_{i+1} \cap L_{i+1}$.

We thus obtain that $\widetilde{p}_h = p_h \cap L_h$, and we derive from item (iii) of Definition 4.1 that $\{q_h\} \times \widetilde{p}_h$ is disjoint from Bad, which contradicts the assumption that $(q_h, \widetilde{p}_h) \in \alpha_{\widetilde{P}}(Bad)$. □

4.2 Parametrized Extrapolation

We propose a generic approach to obtain path invariants by utilizing a parametrized approximation operator for queue contents. The parameter (the k in the definition below) is used to adjust the precision of the approximation.

Definition 4.3. *A* (parametrized) extrapolation *is any function* ∇ *from* $\mathbb{N}$ *to* $\mathcal{R}ec((M^*)^n) \to \mathcal{R}ec((M^*)^n)$ *that satisfies, for any* $L \in \mathcal{R}ec((M^*)^n)$*, the two following conditions (with* $\nabla(k)$ *written as* ∇_k*):*

(i) we have $L \subseteq \nabla_k(L)$ *for every* $k \in \mathbb{N}$*,*
(ii) there exists $k_L \in \mathbb{N}$ *such that* $L = \nabla_k(L)$ *for every* $k \geq k_L$*.*

Our definition of extrapolation is quite general, in particular, it does not require monotonicity in k or in L, but it is adequate for the design of path invariant generation procedures. The most simple extrapolation is the *identity extrapolation* $\lambda k\,.\,(\lambda L\,.\,L)$ that maps each $k \in \mathbb{N}$ to the identity on $\mathcal{R}ec((M^*)^n)$. For example, the parametrized automata approximations of [BHV04] and [LGJJ06] (the latter is in the principal scope of the following discussions and of Appendix B) also satisfy the requirements of Definition 4.3.

Remark 4.4. Extrapolations are closed under various operations, such as functional composition, functional union and intersection, as well as round-robin combination. Formally, for any finite sequence $\nabla^0, \ldots, \nabla^m$ of extrapolations, the functions $\lambda k\,.\,\big(\nabla^0_k \circ \cdots \circ \nabla^m_k\big)$, $\lambda k\,.\,\big(\lambda L\,.\,\bigcup_{i=0}^m \nabla^i_k(L)\big)$ and $\lambda k\,.\,\big(\lambda L\,.\,\bigcap_{i=0}^m \nabla^i_k(L)\big)$ are extrapolations. Moreover, for any infinite sequence $(\mu_k, \nu_k)_{k\in\mathbb{N}}$ of pairs in $\{0, \ldots, m\} \times \mathbb{N}$ such that $(\nu_k)_{k\in\mathbb{N}}$ diverges to infinity, the function $\lambda k\,.\,\nabla^{\mu(k)}_{\nu(k)}$ is an extrapolation. Notice also that any function $f : \mathcal{R}ec((M^*)^n) \to \mathcal{R}ec((M^*)^n)$ that is extensive (i.e., $L \subseteq f(L)$) can be turned into an extrapolation as follows: $\nabla_0 = f$ and $\nabla_k = \lambda L\,.\,L$ for all $k \geq 1$.

4.3 Extrapolation based on Bounded-Depth Bisimulation

We briefly introduce an extrapolation similar to the widening operator introduced in [LGJJ06]. This extrapolation assumes an automata representation of recognizable subsets of $(M^*)^n$, and relies on bounded-depth bisimulation over the states of the automata. For simplicity, we focus on fifo systems with a single queue, i.e., $n = 1$. In this simpler case, recognizable subsets of $(M^*)^n$ are regular languages over M, which can directly be represented by finite automata over M. The general case of $n \geq 2$, which is discussed in detail in Appendices A and B, requires the use of QDDs. More precisely, we consider finite automata over M with a set Q of states. As in abstract regular model checking [BHV04], we use quotienting under equivalence relations on Q to obtain over-approximations of the automaton. However, we follow the approach of [LGJJ06], and focus on bounded-depth bisimulation equivalence.

Given a priori an equivalence relation *col* on Q, also called "coloring", and a bound $k \in \mathbb{N}$, the *colored bisimulation equivalence of depth k* is the equivalence relation $\sim_k^{col}$ on Q defined as: $\sim_0^{col} = col$ and two states are equivalent for $\sim_{k+1}^{col}$ if (1) they are $\sim_k^{col}$-equivalent and (2) they have $\sim_k^{col}$-equivalent m-successors for each letter $m \in M$. The ultimately stationary sequence $\sim_0^{col} \supseteq \sim_1^{col} \supseteq \cdots \supseteq \sim_k^{col} \supseteq \sim_{k+1}^{col} \supseteq \cdots$ of equivalence relations on Q leads to the colored bisimulation-based extrapolation.

We define a coloring *std*, called *standard coloring*, by $(q_1, q_2) \in std$ if either q_1 and q_2 are both final states or q_1 and q_2 are both non-final states. The *bisimulation extrapolation* is the function ρ from $\mathbb{N}$ to $\mathcal{R}ec\,(M^*) \to \mathcal{R}ec\,(M^*)$ defined by $\rho_k(L) = L/\sim_k^{std}$, where L is identified to the minimal deterministic finite automaton accepting it. Notice that ρ is shown to be restricted extrapolation (as introduced later in Definition 4.3). ◇

Remark 4.5. We could also choose other colorings for the above extrapolation or define the sequence of equivalences in a different way. For instance, better results are sometimes obtained in practice with the extrapolation ρ' that first (for $k = 0$) applies a quotienting with respect to the equivalence relation $Q \times Q$ (i.e., all states are merged), and then behaves as ρ_{k-1} (for $k \geq 1$). Analogously, the extrapolation ρ'' defined by $\rho''_0 = \rho'_0$ and $\rho''_k = \rho_k$ for $k \geq 1$ is used in Examples 4.10 and 5.2. The variants ρ' and ρ'' are also formally defined for the general case of $n \geq 2$ in Remark B.10 (page 41).

Example 4.6. Consider the regular language $L = \{aac, baaa\}$ over the alphabet $M = \{a, b, c, d, e\}$, represented by the automaton FA_L of Figure 6a. The previously defined bisimulation-based extrapolation ρ applies to L as follows: ρ_0 splits the states of FA_L according to *std*, hence, $\rho_0(L) = \{a, b, c\}^* \cdot \{a, c\}$ (viz. Figure 6c). Then ρ_1 merges the states that are bisimulation equivalent up to depth 1, i.e., the states ○ of FA_L (Figure 6d). As all states of FA_L are non equivalent for $\sim_k^{std}$ with $k \geq 2$, we have $\rho_k(L) = L$ (again Figure 6a). The variants ρ' and ρ'' mentioned previously would lead to $\rho'_0(L) = \rho''_0(L) = (alph(L))^* = \{a, b, c\}^*$ (viz. Figure 6b). ◇

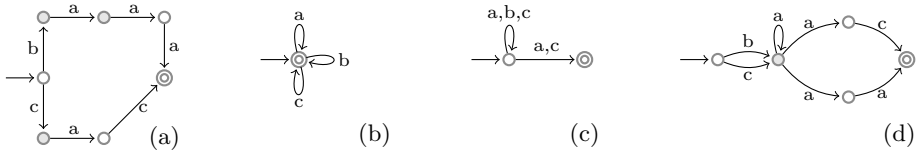


Fig. 6: Finite Automata Representations for Extrapolating L (Example 4.6)

4.4 Extrapolation-based Path Invariant Generation

We now present two extrapolation-based path invariant generation procedures (Figure 7). Recall that the parameter k of an extrapolation intuitively indicates the desired precision of the approximation. The first algorithm, UPInv, performs an approximated *post* computation along the spurious counterexample, and iteratively increases the precision k of the approximation until a path invariant is obtained. The applied precision in UPInv is *uniform* along the counterexample. Due to its simplicity, the termination analysis of CEGAR in Section 6 will refer to UPInv. The second algorithm, APInv, first performs an exact *pre* computation along the spurious counterexample to identify the "bad" coreachable subsets B_i. The path invariant is then computed with a forward traversal that uses the Split subroutine to simplify each *post* image while remaining disjoint from the B_i. The precision used in Split is therefore tailored to each *post* image, which may lead to simpler path invariants. Naturally, both algorithms may be "reversed" to generate path invariants backwards (more precisely, the complement of a path invariant would be generated with the reversed version).

Observe that if the extrapolation ∇ is effectively computable, then all steps in the algorithms UPInv, Split and APInv are effectively computable. We now prove correctness and termination of these algorithms. Let us fix, for the remainder of this section, an extrapolation ∇ and a partition map $P : Q \to \mathbb{P}((M^*)^n)$, and assume that $Init$ and Bad are recognizable.

Proposition 4.7. *For any spurious abstract counterexample $\pi^\sharp$, the execution of* UPInv $(\nabla, Init, Bad, \pi^\sharp)$ *terminates and returns a path invariant for $\pi^\sharp$.*

Proof. Consider a spurious counterexample $\pi^\sharp = (q_0, p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, p_h)$. Let us define the sequence $R_0, \ldots, R_h$ of subsets of $(M^*)^n$ by $R_0 = p_0 \cap \{\boldsymbol{w} \mid (q_0, \boldsymbol{w}) \in Init\}$ and $R_i = post(l_{i-1}, p_{i-1} \cap L_{i-1})$ for all $1 \leq i \leq h$. Notice that $(\{q_h\} \times (p_h \cap R_h)) \cap Bad = \emptyset$ since $\pi^\sharp$ is spurious. According to Definition 4.3, there exists $k_R \in \mathbb{N}$ such that $\nabla_{k_R}(R_i) = R_i$ for every $0 \leq i \leq h$. Consequently, the **while**-loop of the algorithm UPInv (lines 2–11) is re-iterated at most k_R times. Indeed, if $k = k_R$ at some iteration of the **while**-loop, then, for this iteration, $L_i \in \{\emptyset, R_i\}$ for each $0 \leq i \leq h$, and, therefore, $(\{q_h\} \times (p_h \cap L_h)) \cap Bad = \emptyset$. We conclude that the execution of UPInv $(\nabla, Init, Bad, \pi^\sharp)$ terminates.

Let $(L_0, \ldots, L_h)$ denote the value returned by UPInv $(\nabla, Init, Bad, \pi^\sharp)$. It obviously holds that $(\{q_h\} \times (p_h \cap L_h)) \cap Bad = \emptyset$. Recall that, according to

$\mathsf{UPInv}\,(\nabla, Init, Bad, \pi^\sharp)$

Input: extrapolation ∇, recognizable subsets $Init, Bad$ of $Q\times(M^*)^n$, spurious counterexample $\pi^\sharp = (q_0,p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h,p_h)$

```
1  k ← 0
2  do
3     L_0 ← ∇_k (p_0 ∩ {w | (q_0, w) ∈ Init})
4     for i from 1 upto h
5        F_i ← post(l_{i-1}, p_{i-1} ∩ L_{i-1})
6        if p_i ∩ F_i = ∅
7           L_i ← ∅
8        else
9           L_i ← ∇_k(F_i)
10    k ← k + 1
11 while ({q_h} × (p_h ∩ L_h)) ∩ Bad ≠ ∅
12 return (L_0, ..., L_h)
```

$\mathsf{Split}\,(\nabla, L_0, L_1)$

Input: extrapolation ∇, disjoint recognizable subsets L_0, L_1 of $(M^*)^n$

```
1  k ← 0
2  while ∇_k(L_0) ∩ L_1 ≠ ∅
3     k ← k + 1
4  return ∇_k(L_0)
```

$\mathsf{APInv}\,(\nabla, Init, Bad, \pi^\sharp)$

Input: extrapolation ∇, recognizable subsets $Init, Bad$ of $Q\times(M^*)^n$, spurious counterexample $\pi^\sharp = (q_0,p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h,p_h)$

```
1  B_h ← p_h ∩ {w | (q_h, w) ∈ Bad}
2  i ← h
3  while B_i ≠ ∅ and i > 0
4     i ← i − 1
5     B_i ← p_i ∩ pre(l_i, B_{i+1})
6  if B_i ≠ ∅
7     I ← p_0 ∩ {w | (q_0, w) ∈ Init}
8     L_0 ← Split (∇, I, B_0)
9  else
10    (L_0, ..., L_i) ← ((M*)^n, ..., (M*)^n)
11 for j from i + 1 upto h
12    F_j ← post(l_{j-1}, p_{j-1} ∩ L_{j-1})
13    if p_j ∩ F_j = ∅
14       L_j ← ∅
15    else
16       L_j ← Split (∇, F_j, B_j)
17 return (L_0, ..., L_h)
```

Fig. 7: Extrapolation-based Path Invariant Generation Algorithms

Definition 4.3, we have $L \subseteq \nabla_k(L)$ for every $L \in \mathcal{R}ec((M^*)^n)$ and $k \in \mathbb{N}$. We deduce from the definition of the **while**-loop (lines 2–11) that $L_0 \supseteq p_0 \cap \{\boldsymbol{w} \mid (q_0, \boldsymbol{w}) \in Init\}$ and $L_i \supseteq p_i \cap post(l_{i-1}, p_{i-1} \cap L_{i-1})$ for all $1 \leq i \leq h$. We conclude that $(L_0, \ldots, L_h)$ is a path invariant. □

Lemma 4.8. *For any two recognizable subsets L_0, L_1 of $(M^*)^n$, if $L_0 \cap L_1 = \emptyset$ then* $\mathsf{Split}(\nabla, L_0, L_1)$ *terminates and returns a recognizable subset L of $(M^*)^n$ that satisfies $L_0 \subseteq L \subseteq \overline{L_1}$.*

Proof. Consider any two disjoint recognizable subsets L_0, L_1 of $(M^*)^n$. According to Definition 4.3, we have $L = \nabla_{k_L}(L)$ for some $k_L \in \mathbb{N}$, and therefore $\mathsf{Split}(\nabla, L_0, L_1)$ terminates. There exists $k \in \mathbb{N}$ such that the returned value L satisfies $L = \nabla_k(L_0)$ and $\nabla_k(L_0) \cap L_1 = \emptyset$. Since $L_0 \subseteq \nabla_k(L_0)$ from Definition 4.3, we obtain that $L_0 \subseteq L \subseteq \overline{L_1}$. □

Proposition 4.9. *For any spurious abstract counterexample $\pi^\sharp$, the execution of* $\mathsf{APInv}(\nabla, Init, Bad, \pi^\sharp)$ *terminates and returns a path invariant for $\pi^\sharp$.*

Proof. Consider a spurious counterexample $\pi^\sharp = (q_0, p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, p_h)$. Let us define the sequence $R_0, \ldots, R_h$ of subsets of $(M^*)^n$ by $R_h = p_h \cap \{\boldsymbol{w} \mid (q_h, \boldsymbol{w}) \in Bad\}$ and $R_i = p_i \cap pre(l_i, R_{i+1})$ for all $0 \leq i < h$. This sequence satisfies the following disjointness property: for any subset $L \subseteq (M^*)^n$ and for any $0 \leq i < h$, if $L \subseteq \overline{R_i}$ then $post(l_i, p_i \cap L) \subseteq \overline{R_{i+1}}$. Remark that $(\{q_0\} \times R_0) \cap Init = \emptyset$ since $\pi^\sharp$ is spurious.

As the variable i remains nonnegative along the execution of $\mathsf{APInv}(\nabla, Init, Bad, \pi^\sharp)$, the **while**-loop (lines 3–5) and the **for**-loop (lines 11–16) both perform at most h iterations. Hence the execution terminates if each call to Split (lines 8 and 16) terminates. Let us write i_6 the value of the variable i at line 6. Remark that, at line 6 onwards, it holds that $0 \leq i = i_6 \leq h$ and $B_j = R_j$ for each $i_6 \leq j \leq h$. Define $I = p_0 \cap \{\boldsymbol{w} \mid (q_0, \boldsymbol{w}) \in Init\}$. We consider two cases:

- if $B_{i_6} = \emptyset$ then the execution takes the **else** branch (line 10) and therefore $L_j = (M^*)^n$ for all $0 \leq j \leq i_6$ at line 11 onwards. Moreover, as $B_{i_6} = \emptyset$, it holds that $R_j = \emptyset$ for all $0 \leq j \leq i_6$.
- otherwise, $B_{i_6} \neq \emptyset$, which implies that $i_6 = 0$, and the execution proceeds through lines 7–8. Since $B_0 = R_0$, we get that $I \cap B_0 = \emptyset$ at line 8. According to Lemma 4.8, the call to Split at line 8 terminates and L_0 satisfies $I \subseteq L_0 \subseteq \overline{B_0}$.

We obtain in both cases that, before the **for**-loop at line 11, we have: $I \subseteq L_0$ and $p_j \cap post(l_{j-1}, p_{j-1} \cap L_{j-1}) \subseteq L_j \subseteq \overline{R_j}$ for each $0 < j \leq i_6$. We now turn our attention to the **for**-loop (lines 11–16). Consider an iteration $i_6 < j \leq h$ of the **for**-loop, and assume that $L_{j-1} \subseteq \overline{R_{j-1}}$. We deduce from the above mentioned disjointness property that $F_j = post(l_{j-1}, p_{j-1} \cap L_{j-1})$ is disjoint from $R_j = B_j$. Therefore, the call to Split at line 16 (if any) terminates, and, moreover, $p_j \cap post(l_{j-1}, p_{j-1} \cap L_{j-1}) \subseteq L_j \subseteq \overline{R_j}$ at the end of this iteration (using Lemma 4.8 for line 16). We conclude that the execution terminates, and that $I \subseteq L_0$, $L_h \subseteq \overline{R_h}$ and $p_{j+1} \cap post(l_j, p_j \cap L_j) \subseteq L_{j+1}$ for each $0 \leq j < h$, which precisely means that $(L_0, \ldots, L_h)$ is a path invariant. □

Example 4.10. Consider again the c/d protocol, and assume an extrapolation ∇ satisfying $\nabla_0(L \times \varepsilon) = (alph(L))^* \times \varepsilon$ for all $L \subseteq M^*$, and $\nabla_1(u \times \varepsilon) = u \times \varepsilon$ for each $u \in \{\varepsilon, \mathtt{o}, \mathtt{oc}\}$, e.g., the extrapolation ρ'' presented in Remark 4.5. The UPInv algorithm, applied to the spurious counterexample $(\mathtt{00}, \varepsilon \times \varepsilon) \xrightarrow{1!\mathtt{o}}{}^{\sharp} (\mathtt{10}, \mathtt{o}^* \times \varepsilon) \xrightarrow{1!\mathtt{c}}{}^{\sharp} (\mathtt{00}, \overline{\varepsilon \times \varepsilon})$ of Example 3.5, would perform two iterations of the **while**-loop and produce the path invariant $(\varepsilon \times \varepsilon,\ \mathtt{o} \times \varepsilon,\ \mathtt{oc} \times \varepsilon)$. These iterations are detailed in the table below. The marks 👍 and 👎 indicate whether the condition at line 11 is satisfied or not.

	L_0	L_1	L_2	Line 11
$k = 0$	$\varepsilon \times \varepsilon$	$\mathtt{o}^* \times \varepsilon$	$\{\mathtt{o}, \mathtt{c}\}^* \times \varepsilon$	👎
$k = 1$	$\varepsilon \times \varepsilon$	$\mathtt{o} \times \varepsilon$	$\mathtt{oc} \times \varepsilon$	👍

Following Proposition 4.2, the partition map would be refined to:

$q \in Q$	00	10	01, 11
$P(q)$	$\varepsilon \times \varepsilon,\ \mathtt{oc} \times \varepsilon,\ \overline{(\varepsilon \cup \mathtt{oc}) \times \varepsilon}$	$\mathtt{o} \times \varepsilon,\ (\varepsilon \cup (\mathtt{o} \cdot \mathtt{o}^+)) \times \varepsilon,\ \overline{\mathtt{o}^* \times \varepsilon}$	$M^* \times M^*$

This refined partition map clearly rules out the spurious counterexample. ◇

5 Safety Cegar Semi-Algorithm for Fifo Systems

We are now equipped with the key ingredients to present our Cegar semi-algorithm for fifo systems in Figure 8. The semi-algorithm takes as input a fifo system $\mathcal{A}$, a recognizable safety condition $(Init, Bad)$, an initial partition map P_0, and a path invariant generation procedure PathInv. The initial partition map may be the trivial one, mapping each control state to $(M^*)^n$. We may use any path invariant generation procedure, such as the ones presented in the previous section. The semi-algorithm iteratively refines the partition abstraction until either the abstraction is precise enough to prove that $[\![\mathcal{A}]\!]$ is $(Init, Bad)$-safe (line 10), or a feasible counterexample is found (line 4).

The semi-algorithm maintains the current partition map in variable P. At each iteration of the **while**-loop, the partition abstraction $[\![\mathcal{A}]\!]^\sharp_P$ is finite, and any standard graph exploration algorithm may be used to search for an abstract counterexample (lines 1–2). If there is none then $[\![\mathcal{A}]\!]^\sharp_P$ is $(\alpha(Init), \alpha(Bad))$-safe, and the semi-algorithm returns that $[\![\mathcal{A}]\!]$ is $(Init, Bad)$-safe (✓). Otherwise, an arbitrary simple abstract counterexample is chosen depending on an underlying graph search strategy (e.g., breadth-first or depth-first). If this abstract counterexample is feasible then the semi-algorithm returns that $[\![\mathcal{A}]\!]$ is $(Init, Bad)$-unsafe (↯). Or else, a path invariant is generated from the spurious abstract counterexample, and is used to refine the partition. The new partition map obtained after the **foreach** loop (lines 8–9) is precisely the partition map $\widetilde{P}$ from Proposition 4.2, and hence it rules out this abstract counterexample. Recall that

$\mathsf{CEGAR}\,(\mathcal{A}, Init, Bad, P_0, \mathsf{PathInv})$

Input: fifo system $\mathcal{A} = \langle Q, M, n, \Delta\rangle$, recognizable subsets $Init, Bad$ of $Q \times (M^*)^n$, partition map $P_0 : Q \to \mathbb{P}\,((M^*)^n)$, procedure $\mathsf{PathInv}$

1 **while** $[\![\mathcal{A}]\!]_P^\sharp$ is $(\alpha_P(Init), \alpha_P(Bad))$-unsafe
2 pick a simple abstract counterexample $\pi^\sharp$ in $[\![\mathcal{A}]\!]_P^\sharp$
3 **if** $\pi^\sharp$ is a feasible abstract counterexample
4 **return** $\lightning$
5 **else**
6 write $\pi^\sharp$ as the abstract path $(q_0, p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, p_h)$
7 $(L_0, \ldots, L_h) \leftarrow \mathsf{PathInv}\,(Init, Bad, \pi^\sharp)$
8 **foreach** $i \in \{0, \ldots, h\}$
9 $P(q_i) \leftarrow (P(q_i) \setminus \{p_i\}) \cup \left(\{p_i \cap L_i, p_i \cap \overline{L_i}\} \setminus \{\emptyset\}\right)$
10 **return** ✓

Fig. 8: Generic CEGAR Algorithm

Lemmata 3.2 and 3.4 ensure that the steps at lines 1 and 3 are effectively computable.

Let us fix, for the remainder of this section, a fifo system $\mathcal{A}$, two recognizable subsets $Init, Bad$ of $Q \times (M^*)^n$, an initial partition map $P_0 : Q \to \mathbb{P}\,((M^*)^n)$, and a path invariant generation procedure $\mathsf{PathInv}$. The correctness of the CEGAR semi-algorithm is expressed by the following proposition, which directly follows from Proposition 3.3 and from the definition of feasible abstract counterexamples.

Proposition 5.1. *For any terminating execution of* $\mathsf{CEGAR}\,(\mathcal{A}, Init, Bad, P_0, \mathsf{PathInv})$*, if the execution returns* ✓ *(resp.* $\lightning$*) then* $[\![\mathcal{A}]\!]$ *is* $(Init, Bad)$*-safe (resp.* $(Init, Bad)$*-unsafe).*

Example 5.2. We show a full execution of CEGAR on the c/d protocol with initial partition map P_0 defined by $P_0(q) = \{M^* \times M^*\}$ for every $q \in Q$. Recall that $Init = \{(\mathtt{00}, \varepsilon, \varepsilon)\}$ and $Bad = \{\mathtt{00}, \mathtt{10}\} \times (\mathtt{c} \cdot M^* \times M^*)$. Let us assume an extrapolation ∇ that fulfills the following requirements (e.g., the bisimulation-based extrapolation ρ'' previously presented in Remark 4.5):

$$\begin{cases} \nabla_0(L) = (\mathit{alph}\,(\{\boldsymbol{w}(1) \mid \boldsymbol{w} \in L\}))^* \times (\mathit{alph}\,(\{\boldsymbol{w}(2) \mid \boldsymbol{w} \in L\}))^* \\ \nabla_1(u \times \varepsilon) = u \times \varepsilon \text{ for each } u \in \{\varepsilon, \mathtt{o}, \mathtt{oc}, \mathtt{oco}\} \\ \nabla_1(\mathtt{ococ} \times \varepsilon) = (\mathtt{oc})^+ \times \varepsilon \end{cases}$$

We present below the successive iterations of CEGAR. For each iteration, we give the abstract counterexample, the path invariant computed by UPInv, and the refined partition map. We detail the executions of UPInv by providing the potential path invariant at each iteration of the **while**-loop; the marks 👍 and 👎 indicate whether the condition at line 11 of UPInv is satisfied or not. The abstract counterexamples picked at line 2 of CEGAR are obtained by a breadth-first search

of the partition abstraction. All abstract counterexamples are spurious except for the last one which is feasible, and hence the execution of CEGAR returns $\lightning$.

Remark that the partition abstraction obtained at the end of iteration 1 is precisely the one of Example 3.5, and iteration 2 was already presented in Example 4.10.

(0) $\pi^\sharp$ is the empty path $(\mathsf{00}, M^* \times M^*)$.

UPInv:

$k = 0$	$(\varepsilon \times \varepsilon)$	👍

q	$\mathsf{00}$	$\mathsf{10}, \mathsf{01}, \mathsf{11}$
$P(q)$	$\varepsilon \times \varepsilon,\ \overline{\varepsilon \times \varepsilon}$	$M^* \times M^*$

(1) $\pi^\sharp : (\mathsf{00}, \varepsilon \times \varepsilon) \xrightarrow{1!\mathsf{o}}{}^\sharp (\mathsf{10}, M^* \times M^*)$

UPInv:

$k = 0$	$(\varepsilon \times \varepsilon, \mathsf{o}^* \times \varepsilon)$	👍

$\mathsf{00}$	$\mathsf{10}$	$\mathsf{01}, \mathsf{11}$
$\varepsilon \times \varepsilon,\ \overline{\varepsilon \times \varepsilon}$	$\mathsf{o}^* \times \varepsilon,\ \overline{\mathsf{o}^* \times \varepsilon}$	$M^* \times M^*$

(2) $\pi^\sharp : (\mathsf{00}, \varepsilon \times \varepsilon) \xrightarrow{1!\mathsf{o}}{}^\sharp (\mathsf{10}, \mathsf{o}^* \times \varepsilon) \xrightarrow{1!\mathsf{c}}{}^\sharp (\mathsf{00}, \overline{\varepsilon \times \varepsilon})$

UPInv:

$k = 0$	$(\varepsilon \times \varepsilon,\ \mathsf{o}^* \times \varepsilon,\ \{\mathsf{o}, \mathsf{c}\}^* \times \varepsilon)$	👎
$k = 1$	$(\varepsilon \times \varepsilon,\ \mathsf{o} \times \varepsilon,\ \mathsf{oc} \times \varepsilon)$	👍

$\mathsf{00}$	$\mathsf{10}$	$\mathsf{01}, \mathsf{11}$
$\varepsilon \times \varepsilon,\ \mathsf{oc} \times \varepsilon,\ \overline{(\varepsilon \cup \mathsf{oc}) \times \varepsilon}$	$\mathsf{o} \times \varepsilon,\ (\varepsilon \cup (\mathsf{o} \cdot \mathsf{o}^+)) \times \varepsilon,\ \overline{\mathsf{o}^* \times \varepsilon}$	$M^* \times M^*$

(3) $\pi^\sharp : (\mathsf{00}, \varepsilon \times \varepsilon) \xrightarrow{1!\mathsf{o}}{}^\sharp (\mathsf{10}, \mathsf{o} \times \varepsilon) \xrightarrow{1!\mathsf{c}}{}^\sharp (\mathsf{00}, \mathsf{oc}) \xrightarrow{1!\mathsf{o}}{}^\sharp (\mathsf{10}, \overline{\mathsf{o}^* \times \varepsilon})$

UPInv:

$k = 0$	$(\varepsilon \times \varepsilon,\ \mathsf{o}^* \times \varepsilon,\ \{\mathsf{o}, \mathsf{c}\}^* \times \varepsilon,\ \{\mathsf{o}, \mathsf{c}\}^* \times \varepsilon)$	👎
$k = 1$	$(\varepsilon \times \varepsilon,\ \mathsf{o} \times \varepsilon,\ \mathsf{oc} \times \varepsilon,\ \mathsf{oco} \times \varepsilon)$	👍

$\mathsf{00}$	$\varepsilon \times \varepsilon,\ \mathsf{oc} \times \varepsilon,\ \overline{(\varepsilon \cup \mathsf{oc}) \times \varepsilon}$
$\mathsf{10}$	$\mathsf{o} \times \varepsilon,\ (\varepsilon \cup (\mathsf{o} \cdot \mathsf{o}^+)) \times \varepsilon,\ \mathsf{oco} \times \varepsilon,\ \overline{(\mathsf{o}^* \cup \mathsf{oco}) \times \varepsilon}$
$\mathsf{01}, \mathsf{11}$	$M^* \times M^*$

(4) $\pi^\sharp : (\mathsf{00}, \varepsilon \times \varepsilon) \xrightarrow{1!\mathsf{o}}{}^\sharp (\mathsf{10}, \mathsf{o} \times \varepsilon) \xrightarrow{1?\mathsf{o}}{}^\sharp (\mathsf{11}, M^* \times M^*) \xrightarrow{1?\mathsf{c}}{}^\sharp (\mathsf{10}, \overline{(\mathsf{o}^* \cup \mathsf{oco}) \times \varepsilon})$

UPInv:

$k = 0$	$(\varepsilon \times \varepsilon,\ \mathsf{o}^* \times \varepsilon,\ \varepsilon \times \varepsilon,\ \emptyset)$	👍

00	$\varepsilon \times \varepsilon,\ \mathtt{oc} \times \varepsilon,\ \overline{(\varepsilon \cup \mathtt{oc}) \times \varepsilon}$
10	$\mathtt{o} \times \varepsilon,\ (\varepsilon \cup (\mathtt{o} \cdot \mathtt{o}^+)) \times \varepsilon,\ \mathtt{oco} \times \varepsilon,\ \overline{(\mathtt{o}^* \cup \mathtt{oco}) \times \varepsilon}$
01	$M^* \times M^*$
11	$\varepsilon \times \varepsilon,\ \overline{\varepsilon \times \varepsilon}$

(5) $\pi^\sharp : (\mathtt{00}, \varepsilon \times \varepsilon) \xrightarrow{1!\mathtt{o}}{}^\sharp (\mathtt{10}, \mathtt{o} \times \varepsilon) \xrightarrow{1?\mathtt{o}}{}^\sharp (\mathtt{11}, \varepsilon \times \varepsilon) \xrightarrow{2!\mathtt{d}}{}^\sharp (\mathtt{10}, \overline{(\mathtt{o}^* \cup \mathtt{oco}) \times \varepsilon})$

UPInv:

$k = 0$	$(\varepsilon \times \varepsilon,\ \mathtt{o}^* \times \varepsilon,\ \varepsilon \times \varepsilon,\ \varepsilon \times \mathtt{d}^*)$	👍

00	$\varepsilon \times \varepsilon,\ \mathtt{oc} \times \varepsilon,\ \overline{(\varepsilon \cup \mathtt{oc}) \times \varepsilon}$
10	$\mathtt{o} \times \varepsilon,\ (\varepsilon \cup (\mathtt{o} \cdot \mathtt{o}^+)) \times \varepsilon,\ \mathtt{oco} \times \varepsilon,\ \varepsilon \times \mathtt{d}^+,\ \overline{((\mathtt{o}^* \cup \mathtt{oco}) \times \varepsilon) \cup (\varepsilon \times \mathtt{d}^+)}$
01	$M^* \times M^*$
11	$\varepsilon \times \varepsilon,\ \overline{\varepsilon \times \varepsilon}$

(6) $\pi^\sharp : \underline{(\mathtt{00}, \varepsilon \times \varepsilon)} \xrightarrow{1!\mathtt{o}}{}^\sharp (\mathtt{10}, \mathtt{o} \times \varepsilon) \xrightarrow{1!\mathtt{c}}{}^\sharp (\mathtt{00}, \mathtt{oc} \times \varepsilon) \xrightarrow{1!\mathtt{o}}{}^\sharp (\mathtt{10}, \mathtt{oco} \times \varepsilon) \xrightarrow{1!\mathtt{c}}{}^\sharp (\mathtt{00}, \overline{(\varepsilon \cup \mathtt{oc}) \times \varepsilon})$

UPInv:

$k = 0$	$(\varepsilon \times \varepsilon,\ \mathtt{o}^* \times \varepsilon,\ \{\mathtt{o}, \mathtt{c}\}^* \times \varepsilon,\ \{\mathtt{o}, \mathtt{c}\}^* \times \varepsilon,\ \{\mathtt{o}, \mathtt{c}\}^* \times \varepsilon)$	👎
$k = 1$	$(\varepsilon \times \varepsilon,\ \mathtt{o} \times \varepsilon,\ \mathtt{oc} \times \varepsilon,\ \mathtt{oco} \times \varepsilon,\ (\mathtt{oc})^+ \times \varepsilon)$	👍

00	$\varepsilon \times \varepsilon,\ \mathtt{oc} \times \varepsilon,\ (\mathtt{oc} \cdot (\mathtt{oc})^+) \times \varepsilon,\ \overline{(\mathtt{oc})^* \times \varepsilon}$
10	$\mathtt{o} \times \varepsilon,\ (\varepsilon \cup (\mathtt{o} \cdot \mathtt{o}^+)) \times \varepsilon,\ \mathtt{oco} \times \varepsilon,\ \varepsilon \times \mathtt{d}^+,\ \overline{((\mathtt{o}^* \cup \mathtt{oco}) \times \varepsilon) \cup (\varepsilon \times \mathtt{d}^+)}$
01	$M^* \times M^*$
11	$\varepsilon \times \varepsilon,\ \overline{\varepsilon \times \varepsilon}$

(7) $\pi^\sharp : \underline{(\mathtt{00}, \varepsilon \times \varepsilon)} \xrightarrow{1!\mathtt{o}}{}^\sharp (\mathtt{10}, \mathtt{o} \times \varepsilon) \xrightarrow{1!\mathtt{c}}{}^\sharp (\mathtt{00}, \mathtt{oc} \times \varepsilon) \xrightarrow{1?\mathtt{o}}{}^\sharp (\mathtt{01}, M^* \times M^*) \xrightarrow{2!\mathtt{d}}{}^\sharp (\mathtt{00}, \overline{(\mathtt{oc})^* \times \varepsilon})$. This abstract counterexample is feasible. ◇

Remark 5.3. The general benefits of the bisimulation extrapolation (already introduced in Section 4.3) for the abstraction of fifo systems were already discussed in [LGJJ06]. The above iteration (6) of the CEGAR algorithm shows that this extrapolation can, in some common cases, discover *exact* repetitions of message sequences in queues, without the need for additional acceleration techniques.

Let us consider the first queue only. The application of acceleration techniques on the path $(\mathtt{00}, \varepsilon) \xrightarrow{1!\mathtt{o}} \xrightarrow{1!\mathtt{c}} (\mathtt{00}, \mathtt{oc}) \xrightarrow{1!\mathtt{o}} \xrightarrow{1!\mathtt{c}} (\mathtt{00}, \mathtt{ococ}) \cdots$ produces the set of queue contents $(\mathtt{oc})^+$. The bisimulation extrapolation ρ applied to the singleton language $\{\mathtt{ococ}\}$, represented by the obvious automaton, produces the following results for the first two parameters: $\rho_0(\{\mathtt{ococ}\}) = \{\mathtt{o}, \mathtt{c}\}^* \cdot \mathtt{c}$ and $\rho_1(\{\mathtt{ococ}\}) = (\mathtt{oc})^+$.

Termination of the CEGAR semi-algorithm cannot be assured as, otherwise, it would solve the general reachability problem (given any two configurations c

and c' in $\mathcal{C}$, decide whether there exists a path from c to c'), which is known to be undecidable for fifo systems [BZ83]. However, $(Init, Bad)$-unsafety is semi-decidable for fifo systems by forward or backward symbolic exploration when $Init$ and Bad are recognizable [BG99]. Moreover, this problem becomes decidable for fifo systems having a finite reachability set from $Init$.

6 Termination Analysis of the CEGAR Semi-Algorithm

We investigate in this section the termination of the CEGAR semi-algorithm when $\mathcal{A}$ is $(Init, Bad)$-unsafe or has a finite reachability set from $Init$. In contrast to other approaches where abstractions are refined globally (e.g., predicate abstraction [GS97]), partition abstractions [CGJ+03] are refined locally by splitting abstract configurations along the abstract counterexample (viz. lines 8 – 9 of the CEGAR semi-algorithm). The abstract transition relation only needs to be refined locally around the abstract configurations which have been split, and, hence, its refinement can be computed efficiently. However, this local nature of refinement complicates the analysis of the algorithm.

6.1 Characterization of Non-Terminating Executions of CEGAR

First we introduce some additional notations. For any set $\mathcal{L}$ of subsets of $(M^*)^n$, we denote by $\Psi(\mathcal{L})$ the set of equivalence classes of the equivalence relation $\sim_{\mathcal{L}}$ on $(M^*)^n$ defined by: $\boldsymbol{w} \sim_{\mathcal{L}} \boldsymbol{w'}$ if for every $L \in \mathcal{L}$, we have $\boldsymbol{w} \in L$ if and only if $\boldsymbol{w'} \in L$. Intuitively, $\Psi(\mathcal{L})$ is the partition "generated" by $\mathcal{L}$. Notice that if $\mathcal{L}$ is finite then so is $\Psi(\mathcal{L})$.

Given an execution of CEGAR $(\mathcal{A}, Init, Bad, P_0, \mathsf{PathInv})$, and for each iteration $\theta \in \{0, 1, 2, \ldots\}$ of the **while**-loop (only considering iterations that do not return at line 4), we take a "snapshot" between lines 7 and 8, and remember the current partition map as P_θ, the simple abstract counterexample as $\pi^\sharp_\theta$ and its length as h_θ, and the path invariant as $(L^\theta_0, \ldots, L^\theta_{h_\theta})$. Moreover we shortly write $[\![\mathcal{A}]\!]^\sharp_\theta$, $\mathcal{C}^\sharp_\theta$ and α_θ instead of $[\![\mathcal{A}]\!]^\sharp_{P_\theta}$, $\mathcal{C}^\sharp_{P_\theta}$ and α_{P_θ}, respectively. We also define $Init^\sharp_\theta = \alpha_\theta(Init)$ and $Bad^\sharp_\theta = \alpha_\theta(Bad)$. For any bound $b \in \mathbb{N}$, we let $Reach^{\leq b}_\theta$ denote the set of abstract configurations $(q, p) \in \mathcal{C}^\sharp_\theta$ such that there exists in $[\![\mathcal{A}]\!]^\sharp_\theta$ a path of length at most b from $Init^\sharp_\theta$ to (q, p).

Lemma 6.1. *Consider any execution of* CEGAR $(\mathcal{A}, Init, Bad, P_0, \mathsf{PathInv})$. *For any iteration θ and for any $(q, p) \in \mathcal{C}^\sharp_\theta$, it holds that $p \in \Psi(\mathcal{L}_\theta(q))$ where:*

$$\mathcal{L}_\theta(q) \;=\; P_0(q) \cup \{L^\eta_i \mid 0 \leq \eta < \theta \text{ and } 0 \leq i \leq h_\eta\}$$

Proof. We prove the lemma by induction on θ. The basis is trivial, since $p \in P_0(q)$ for every $(q, p) \in \mathcal{C}^\sharp_0$. Assume that the lemma holds for the iteration θ and let us show that the lemma also holds for the iteration $\theta+1$. Let $(q, p) \in \mathcal{C}^\sharp_{\theta+1}$. If $(q, p) \in \mathcal{C}^\sharp_\theta$, then we get that $p \in \Psi(\mathcal{L}_\theta(q))$. Since $\mathcal{L}_\theta(q) \subseteq \mathcal{L}_{\theta+1}(q)$, we obtain that $p \in$

$\Psi(\mathcal{L}_{\theta+1}(q))$. Assume now that $(q,p) \notin \mathcal{C}^\sharp_\theta$. Since $p \in P_{\theta+1}(q) \setminus P_\theta(q)$, we get that p was added to $P(q)$ during the iteration θ at line 9. We deduce from line 9 that $p \in \left\{p' \cap L, p' \cap \overline{L}\right\}$ for some $(q',p') \in \mathcal{C}^\sharp_\theta$ and $L \in \left\{L^\theta_i \;\middle|\; 0 \leq i \leq h_\theta\right\}$. We deduce from the induction hypothesis $p' \in \Psi(\mathcal{L}_\theta(q))$ and therefore $p \in \Psi(\mathcal{L}_{\theta+1}(q))$. □

Proposition 6.2. *For any non-terminating execution of* $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{PathInv})$*, the set* $\left\{L^\theta_i \;\middle|\; \theta \in \mathbb{N} \text{ and } 0 \leq i \leq h_\theta\right\}$ *is infinite.*

Proof. Consider a non-terminating execution and let us show that the set $\mathcal{L} = \left\{L^\theta_i \;\middle|\; \theta \in \mathbb{N}, 0 \leq i \leq h_\theta\right\}$ is infinite. We get from Lemma 6.1 that for every $q \in Q$ and $\theta \in \mathbb{N}$, we have $P_\theta(q) \subseteq \Psi(P_0(q) \cup \mathcal{L})$. According to line 9 of the CEGAR semi-algorithm, $P_{\theta+1}$ refines P_θ for every $\theta \in \mathbb{N}$, and moreover $P_{\theta+1} \neq P_\theta$ since $P_{\theta+1}$ rules out $\pi^\sharp_\theta$. We deduce that there exists $q \in Q$ such that the nondecreasing sequence $(|P_\theta(q)|)_{\theta\in\mathbb{N}}$ diverges. Since $P_0(q)$ is finite and $P_\theta(q) \subseteq \Psi(P_0(q) \cup \mathcal{L})$, we conclude that $\mathcal{L}$ is infinite. □

6.2 Properties of Breadth-First Executions of CEGAR

To obtain termination results in the unsafe case, we will, unsurprisingly, restrict ourselves to breadth-first explorations of the partition abstractions. Formally, a *breadth-first* execution of the CEGAR semi-algorithm is any execution where, at each iteration θ, the abstract counterexample $\pi^\sharp_\theta$ picked at line 2 is among the shortest ones.

Lemma 6.3. *Consider any breadth-first execution of* $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{PathInv})$*. For any iteration* $\theta \geq 1$ *and for any* $(q,p) \in Init^\sharp_\theta \setminus Init^\sharp_{\theta-1}$*, there exists* $p_0 \in P_{\theta-1}(q)$ *such that* $p = p_0 \cap L_0^{\theta-1}$ *and* $(\{q\} \times p_0) \cap Init = (\{q\} \times p) \cap Init$*.*

Proof. Consider an iteration $\theta+1$ (with $\theta \in \mathbb{N}$) and let (q,p) be any abstract configuration in $Init^\sharp_{\theta+1} \setminus Init^\sharp_\theta$. Observe that $(\{q\} \times p) \cap Init$ is non-empty, and therefore $(q,p) \notin \mathcal{C}^\sharp_\theta$ since otherwise we would have $(q,p) \in Init^\sharp_\theta$. Since $p \in P_{\theta+1}(q) \setminus P_\theta(q)$, we get that p was added to $P(q)$ during the iteration θ at line 9. Let us write $\pi^\sharp_\theta$ as $\pi^\sharp_\theta = (q_0,p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h,p_h)$. We deduce from line 9 that $p \in \left\{p_{i_0} \cap L^\theta_{i_0}, p_{i_0} \cap \overline{L^\theta_{i_0}}\right\}$ for some $0 \leq i_0 \leq h$ such that $q = q_{i_0}$. Observe that $(q_{i_0}, p_{i_0}) \in Init^\sharp_\theta$ since we have $p \subseteq p_{i_0}$ and $(\{q\} \times p) \cap Init \neq \emptyset$. We come to $i_0 = 0$ since the abstract counterexample $\pi^\sharp_\theta$ is among the shortest ones. Hence, we get that $q = q_0$ and $p \in \left\{p_0 \cap L^\theta_0, p_0 \cap \overline{L^\theta_0}\right\}$. Since $(L^\theta_0, \ldots, L^\theta_h)$ is a path invariant for $\pi^\sharp_\theta$, we have $(\{q_0\} \times p_0) \cap Init \subseteq \{q_0\} \times L^\theta_0$ and hence $\{q_0\} \times (p_0 \cap \overline{L^\theta_0})$ is disjoint from $Init$. We deduce that $p = p_0 \cap L^\theta_0$ and moreover we get that $(\{q\} \times p_0) \cap Init = (\{q\} \times p) \cap Init$. □

Lemma 6.4. *Consider any breadth-first execution of* $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{PathInv})$*. For any iteration* θ*, for any* $b \in \mathbb{N}$ *and for any* $(q,p) \in Reach^{\leq b}_\theta$*, we have* $p \in \Psi\left(\mathcal{L}^b_\theta(q)\right)$ *where:*

$$\mathcal{L}^b_\theta(q) \;=\; P_0(q) \cup \{L^\eta_i \mid 0 \leq \eta < \theta, i \leq h_\theta \text{ and } i \leq b\}$$

Proof. For any iteration θ and for any $b \in \mathbb{N}$, let use denote by (H^b_θ) the property: for any $(q,p) \in \mathcal{C}^\sharp_\theta$, if there exists in $[\![\mathcal{A}]\!]^\sharp_\theta$ a path of length at most b from $Init^\sharp_\theta$ to (q,p), then $p \in \Psi\big(\mathcal{L}^b_\theta(q)\big)$. We prove by double induction on θ and b that (H^b_θ) holds for any iteration θ and for any $b \in \mathbb{N}$.

Let us prove the basis $\forall b\,(H^b_0)$ of the induction on θ. Observe that $\mathcal{L}^b_0(q) = P_0(q)$ for every $q \in Q$. Therefore $p \in P_0(q) = \Psi(P_0(q))$ for any $(q,p) \in \mathcal{C}^\sharp_0$, and we conclude that the basis obviously holds. We now prove the induction step $\forall\theta\,\big(\forall b\,(H^b_\theta) \implies \forall b\,(H^b_{\theta+1})\big)$ of the induction on θ. Consider an iteration $\theta+1$ (with $\theta \in \mathbb{N}$) and assume that (H^b_θ) holds for every $b \in \mathbb{N}$. We prove by induction on b that $(H^b_{\theta+1})$ holds for any $b \in \mathbb{N}$. Observe that the basis $(H^0_{\theta+1})$ may equivalently be rephrased as: for any $(q,p) \in Init^\sharp_{\theta+1}$, we have $p \in \Psi\big(\mathcal{L}^0_{\theta+1}\big)$. Let $(q,p) \in Init^\sharp_{\theta+1}$. If $(q,p) \in Init^\sharp_\theta$ then we deduce from (H^0_θ) that $p \in \Psi\big(\mathcal{L}^0_\theta\big)$. Since $\mathcal{L}^0_\theta \subseteq \mathcal{L}^0_{\theta+1}$ we obtain that $p \in \Psi\big(\mathcal{L}^0_{\theta+1}\big)$. Otherwise, we obtain from Lemma 6.3 that $p = p_0 \cap L^\theta_0$ for some $p_0 \in P_\theta(q)$. We deduce from (H^0_θ) that $p_0 \in \Psi\big(\mathcal{L}^0_\theta\big)$ and therefore $p \in \Psi\big(\mathcal{L}^0_{\theta+1}\big)$. We therefore have proved that the basis $(H^0_{\theta+1})$ of the induction on b holds.

Let us now show the induction step $\forall b\,\big((H^b_{\theta+1}) \implies (H^{b+1}_{\theta+1})\big)$ of the induction on b. Consider any bound $b \in \mathbb{N}$ and assume that $(H^b_{\theta+1})$ holds. Recall that (H^c_θ) holds for every $c \in \mathbb{N}$. Let (q,p) be any abstract configuration in $\mathcal{C}^\sharp_{\theta+1}$ such that there is in $[\![\mathcal{A}]\!]^\sharp_{\theta+1}$ a path $\pi^\sharp$ of length at most $b+1$ from $Init^\sharp_{\theta+1}$ to (q,p). We show that $p \in \Psi\big(\mathcal{L}^{b+1}_{\theta+1}(q)\big)$. Recall that $P_{\theta+1}(q)$ refines $P_\theta(q)$ and define $\hat{p} = [\,p\,]_{P_\theta(q)}$, i.e. $\hat{p}$ is the class in $P_\theta(q)$ that contains p. Observe that $(q,\hat{p})$ is an abstract configuration in $\mathcal{C}^\sharp_\theta$. The "lift" of $\pi^\sharp$ to P_θ yields a path of length at most $b+1$ in $[\![\mathcal{A}]\!]^\sharp_\theta$ from $Init^\sharp_\theta$ to $(q,\hat{p})$. We deduce from (H^{b+1}_θ) that $\hat{p} \in \Psi\big(\mathcal{L}^{b+1}_\theta(q)\big)$. Since $\mathcal{L}^{b+1}_\theta \subseteq \mathcal{L}^{b+1}_{\theta+1}$ we obtain that $\hat{p} \in \Psi\big(\mathcal{L}^{b+1}_{\theta+1}\big)$. If $p \in P_\theta(q)$ then $p = \hat{p} \in \Psi\big(\mathcal{L}^{b+1}_\theta(q)\big)$. Otherwise, $p \in P_{\theta+1}(q) \setminus P_\theta(q)$ and we get that p was added to $P(q)$ during the iteration θ at line 9. Let us write $\pi^\sharp_\theta$ as $\pi^\sharp_\theta = (q_0,p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h,p_h)$. We deduce from line 9 that $p \in \left\{p_{i_0} \cap L^\theta_{i_0}, p_{i_0} \cap \overline{L^\theta_{i_0}}\right\}$ for some $0 \le i_0 \le h$ such that $q = q_{i_0}$. Moreover $p_{i_0} = \hat{p}$ since p_{i_0} and $\hat{p}$ both contain p. Remark that we may replace in $\pi^\sharp_\theta$ the prefix $(q_0,p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{i_0-1}}{}^\sharp (q_{i_0},\hat{p})$ with the "lift" of $\pi^\sharp$ to P_θ. The resulting abstract path is also an abstract counterexample in $[\![\mathcal{A}]\!]^\sharp_\theta$, and its length is $h - i_0 + (b+1)$. Since $\pi^\sharp_\theta$ is among the shortest ones, we get that $i_0 \le b+1$. As $p_{i_0} = \hat{p} \in \Psi\big(\mathcal{L}^{b+1}_\theta(q)\big)$, we conclude that $p \in \Psi\big(\mathcal{L}^{b+1}_\theta(q)\big)$. □

Lemma 6.5. *Consider any breadth-first execution of* $\mathsf{CEGAR}\,(\mathcal{A}, Init, Bad, P_0, \mathsf{PathInv})$*, and define* $\mathcal{I}_\theta = \left\{ p \cap \{\boldsymbol{w} \mid (q,\boldsymbol{w}) \in Init\} \;\middle|\; (q,p) \in Init^\sharp_\theta \right\}$ *for any iteration* θ*. It holds that* $\mathcal{I}_\theta \subseteq \mathcal{I}_{\theta-1}$ *for any iteration* $\theta \ge 1$*.*

Proof. Consider an iteration $\theta \ge 1$ and let $L \in \mathcal{I}_\theta$. There exists $(q,p) \in Init^\sharp_\theta$ such that $L = p \cap \{\boldsymbol{w} \mid (q,\boldsymbol{w}) \in Init\}$. Notice that $\{q\} \times L = (\{q\} \times p) \cap Init \neq \emptyset$. If $(q,p) \in Init^\sharp_{\theta-1}$ then $L \in \mathcal{I}_{\theta-1}$. Otherwise, we obtain from Lemma 6.3 that

$(\{q\} \times p_0) \cap Init = (\{q\} \times p) \cap Init$ for some $p_0 \in P_{\theta-1}(q)$. We thus come to $L = p_0 \cap \{\boldsymbol{w} \mid (q, \boldsymbol{w}) \in Init\}$. Since $L \neq \emptyset$, we get that $(q, p_0) \in Init^\sharp_{\theta-1}$ and we conclude that $L \in \mathcal{I}_{\theta-1}$. □

6.3 Termination of CEGAR for Unsafe fifo Systems with UPInv (∇)

To our knowledge, there is currently no *general* result on the termination of CEGAR for infinite transition systems that are unsafe. In the following, we present an answer for a particular setting with respect to our case.

For the rest of this section, we fix an extrapolation ∇ and we focus on the path invariant generation procedure UPInv presented in Section 4.

The following proposition shows that for any bound b, there is an iteration after which the abstract configurations that are reachable from $Init^\sharp$ by a path of length at most b are never split, or, put differently, the "reachability set up to depth b" of the abstraction remains constant.

Proposition 6.6. *For any $b \in \mathbb{N}$ and for any non-terminating breadth-first execution of* CEGAR $(\mathcal{A}, Init, Bad, P_0,$ UPInv $(\nabla))$*, the two following sets are finite:*

$$\bigcup_{\theta \in \mathbb{N}} Reach^{\leq b}_\theta \qquad \text{and} \qquad \left\{ L^\theta_i \;\middle|\; \theta \in \mathbb{N}, i \leq h_\theta \text{ and } i \leq b \right\}$$

Proof. We prove the proposition by induction on b. Let us first show the basis. For any $\theta \in \mathbb{N}$, define $\mathcal{I}_\theta$ as in Lemma 6.5. We infer from Lemma 6.5 that $\mathcal{I}_\theta \subseteq \mathcal{I}_0$. We derive from the definition of the algorithm UPInv that for any iteration $\theta \in \mathbb{N}$, there exists $(q, p) \in Init^\sharp_\theta$ and $k \in \mathbb{N}$ such that $L^\theta_0 = \nabla_k (p \cap \{\boldsymbol{w} \mid (q, \boldsymbol{w}) \in Init\})$, and therefore $L^\theta_0 = \nabla_k(L)$ for some $L \in \mathcal{I}_\theta$. Recall that according to Definition 4.3, the set $\{\nabla_k(L) \mid k \in \mathbb{N}\}$ is finite for any recognizable subset L of $(M^*)^n$. Since $\mathcal{I}_0$ is finite, we obtain that $\{\nabla_k(L) \mid L \in \mathcal{I}_0, k \in \mathbb{N}\}$ is finite. Consequently, the set $\left\{L^\theta_0 \mid \theta \in \mathbb{N}\right\}$ is finite. Moreover, according to Lemma 6.4, we have $p \in \Psi\big(P_0(q) \cup \left\{L^\theta_0 \mid \theta \in \mathbb{N}\right\}\big)$ for every $(q, p) \in Reach^{\leq 0}_\theta$. We deduce that $\bigcup_{\theta \in \mathbb{N}} Reach^{\leq 0}_\theta$ is finite.

Let us now show the induction step. Assume that the proposition holds for some bound $b \in \mathbb{N}$. Let us define $\mathcal{H} = \left\{ p \cap L^\theta_b \;\middle|\; \theta \in \mathbb{N}, b \leq h_\theta, (q, p) \in Reach^{\leq b}_\theta \right\}$. The sets $\bigcup_{\theta \in \mathbb{N}} Reach^{\leq b}_\theta$ and $\left\{L^\theta_b \mid \theta \in \mathbb{N}, b \leq h_\theta\right\}$ are both finite according to the induction hypothesis, and therefore $\mathcal{H}$ is finite. We derive from the definition of the algorithm UPInv that for any iteration $\theta \in \mathbb{N}$ with $h_\theta \geq b + 1$, if L^θ_{b+1} is non-empty then there exists $(q, p) \in Reach^{\leq b}_\theta$, $l \in \Sigma$ and $k \in \mathbb{N}$ such that $L^\theta_{b+1} = \nabla_k \left(post(l, p \cap L^\theta_b)\right)$, and therefore $L^\theta_{b+1} = \nabla_k \left(post(l, L)\right)$ for some $L \in \mathcal{H}$. Recall that according to Definition 4.3, the set $\{\nabla_k (L) \mid k \in \mathbb{N}\}$ is finite for any subset L of $(M^*)^n$. Since $\mathcal{H}$ and Σ are both finite, we obtain that $\{\nabla_k \left(post(l, L)\right) \mid l \in \Sigma, L \in \mathcal{H}, k \in \mathbb{N}\}$ is finite. We deduce that the set $\left\{L^\theta_{b+1} \mid \theta \in \mathbb{N}, b + 1 \leq h_\theta\right\}$ is finite, and we get from the induction hypothesis that $\left\{L^\theta_i \mid \theta \in \mathbb{N}, i \leq h_\theta, i \leq b + 1\right\}$ is also finite. Moreover, according to Lemma 6.4, we have $p \in \Psi\big(P_0(q) \cup \left\{L^\theta_i \mid \theta \in \mathbb{N}, i \leq h_\theta, i \leq b + 1\right\}\big)$ for every $(q, p) \in Reach^{\leq b+1}_\theta$. We deduce that $\bigcup_{\theta \in \mathbb{N}} Reach^{\leq b+1}_\theta$ is finite. □

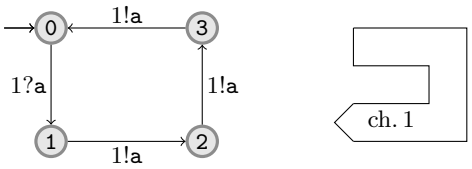


Fig. 9: Fifo System of Example 6.9 Showing Non-Termination of CEGAR

Proposition 6.7. *For any breadth-first execution of* $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$*, if the execution does not terminate then the sequence* $(h_\theta)_{\theta\in\mathbb{N}}$ *of lengths of counterexamples picked at line 2 is nondecreasing and diverges.*

Proof. Consider a non-terminating breadth-first execution and let us show that the sequence $(h_\theta)_{\theta\in\mathbb{N}}$ is nondecreasing and diverges. Let $\eta, \theta \in \mathbb{N}$ such that $\eta < \theta$, and observe that the partition map P_θ refines P_η. The "lift" of $\pi^\sharp_\theta$ to P_η yields a counterexample in $[\![\mathcal{A}]\!]^\sharp_\eta$. Since $\pi^\sharp_\eta$ is a counterexample in $[\![\mathcal{A}]\!]^\sharp_\eta$ among the shortest ones, we get that its length h_η satisfies $h_\eta \leq h_\theta$. This concludes the proof that $(h_\theta)_{\theta\in\mathbb{N}}$ is nondecreasing.

By contradiction, assume that there exists $b, \theta_1 \in \mathbb{N}$ such that $h_\theta = b$ for every $\theta \geq \theta_1$. We obtain from Proposition 6.6 that $\bigcup_{\theta\in\mathbb{N}} Reach^{\leq b}_\theta$ is finite. Therefore, there exists $\theta_2 \geq \theta_1$ such that $Reach^{\leq b}_{\theta_2} = Reach^{\leq b}_{\theta_2+1}$. Let us write $\pi^\sharp_{\theta_2}$ as $\pi^\sharp_{\theta_2} = (q_0, p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{b-1}}{}^\sharp (q_b, p_b)$. Note that $(q_i, p_i) \in Reach^{\leq b}_{\theta_2+1}$ for every $0 \leq i \leq b$. We deduce that $\pi^\sharp_{\theta_2}$ is also a counterexample in $[\![\mathcal{A}]\!]^\sharp_{\theta_2+1}$, which contradicts the fact that P_{θ_2+1} is a refinement of P_{θ_2} that rules out $\pi^\sharp_{\theta_2}$. We conclude that the sequence $(h_\theta)_{\theta\in\mathbb{N}}$ diverges. □

Corollary 6.8. *If* $[\![\mathcal{A}]\!]$ *is* $(Init, Bad)$*-unsafe then any breadth-first execution of* $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$ *terminates.*

Proof. Assume that there exists in $[\![\mathcal{A}]\!]$ a path π from $Init$ to Bad and let b denote the length of π. Consider any breadth-first execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$. Observe that for any iteration θ, $\alpha_\theta(\pi)$ is an abstract counterexample of length b in $[\![\mathcal{A}]\!]^\sharp_\theta$. Hence, we have $h_\theta \leq b$ for every iteration $\theta \in \mathbb{N}$, and we conclude with Proposition 6.7 that the execution terminates. □

6.4 Termination of CEGAR for Finite fifo Systems with UPInv (∇)

It would also be desirable to obtain termination of the CEGAR semi-algorithm when $\mathcal{A}$ has a finite reachability set from $Init$. However, the following example shows that this condition is not sufficient to guarantee that $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$ has a terminating execution.

Example 6.9. Consider the fifo system $\mathcal{A}$ depicted in Figure 9. This fifo system has a single message a and a single queue. The safety condition $(Init, Bad)$ is defined by the recognizable subsets $Init = \{(0, \varepsilon)\}$ and $Bad = \{0\} \times (\{a\} \cdot \{aa\}^*)$. Notice that the reachability set from $Init$ is equal to $Init$, which is finite when the initial set of configurations is given as before, and hence $[\![\mathcal{A}]\!]$ is $(Init, Bad)$-safe.

Define the initial partition map P_0 by $P_0(q) = \{\{a\}^*\}$ for all $q \in \{0, 1, 2, 3\}$. We consider the extrapolation ∇ defined by $\nabla_0(\{\varepsilon\}) = \{\varepsilon, aa\}$ and $\nabla_k(L) = L$ if $k > 0$ or $L \neq \{\varepsilon\}$. Let us now detail the first iterations of an execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$.

(0) $\pi_0^\sharp$ is the empty path $(0, \{a\}^*)$, and $\mathsf{UPInv}(\nabla, Init, Bad, \pi_0^\sharp)$ returns the path invariant $(\{\varepsilon, aa\})$.

(1) $\pi_1^\sharp$ is the path $(0, \{\varepsilon, aa\}) \xrightarrow{1?a}{}^\sharp (1, \{a\}^*) \xrightarrow{1!a}{}^\sharp (2, \{a\}^*) \xrightarrow{1!a}{}^\sharp (3, \{a\}^*) \xrightarrow{1!a}{}^\sharp (0, \overline{\{\varepsilon, aa\}})$, and the path invariant is $(\{\varepsilon, aa\}, \{a\}, \{aa\}, \{a^3\}, \{a^4\})$.

(2) $\pi_2^\sharp$ is the path $(0, \{\varepsilon, aa\}) \xrightarrow{1?a}{}^\sharp (1, \{a\}) \xrightarrow{1!a}{}^\sharp (2, \{aa\}) \xrightarrow{1!a}{}^\sharp (3, \{a^3\}) \xrightarrow{1!a}{}^\sharp (0, \{a^4\}) \xrightarrow{1?a}{}^\sharp (1, \overline{\{a\}}) \xrightarrow{1!a}{}^\sharp (2, \overline{\{aa\}}) \xrightarrow{1!a}{}^\sharp (3, \overline{\{a^3\}}) \xrightarrow{1!a}{}^\sharp (0, \overline{\{\varepsilon, aa, a^4\}})$, and the path invariant returned by $\mathsf{UPInv}(\nabla, Init, Bad, \pi_2^\sharp)$ is the sequence $(\{\varepsilon, aa\}, \{a\}, \{a^2\}, \{a^3\}, \{a^4\}, \{a^3\}, \{a^4\}, \{a^5\}, \{a^6\})$.

These first iterations suggest that the execution may not terminate, and we can actually prove that it necessarily does not terminate. Consider any execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$. For any iteration θ, the path invariant $(L_0^\theta, \ldots, L_{h_\theta}^\theta)$ computed by $\mathsf{UPInv}(\nabla)$ satisfies $L_0^\theta = \{\varepsilon, aa\}$ and $L_{4i}^\theta = \{a^4 \cdot a^{2(i-1)}\}$ for any $1 \leq i \leq \frac{h_\theta}{4}$. We deduce that, for each iteration θ, there exists a finite subset F_θ of $\{a\}^*$ such that $\{\{\varepsilon, aa\}, \overline{F_\theta}\} \subseteq P_\theta(0)$. Observe that $(0, \{\varepsilon, aa\}) \in Init_\theta^\sharp$ and $(0, \overline{F_\theta}) \in Bad_\theta^\sharp$. Moreover, for every $i \geq 1$, there is a concrete path in $[\![\mathcal{A}]\!]$ from $(0, aa)$ to $(0, a^{2i})$. Hence, there is an abstract path in $[\![\mathcal{A}]\!]_\theta^\sharp$ from $(0, \{\varepsilon, aa\})$ to $(0, \overline{F_\theta})$. We obtain that $[\![\mathcal{A}]\!]_\theta^\sharp$ is $(Init^\sharp, Bad^\sharp)$-unsafe for every iteration θ, which, combined with Proposition 5.1, implies that the execution does not terminate since $[\![\mathcal{A}]\!]$ is $(Init, Bad)$-safe. ◇

It turns out that termination of the CEGAR semi-algorithm can be guaranteed for fifo systems with a finite reachability set when ∇_k has a finite image for every $k \in \mathbb{N}$. This apparently strong requirement, formally specified in Definition 6.10, is satisfied by the extrapolations presented in [BHV04] and [LGJJ06], which are based on state equivalences up to a certain depth (see Appendix B).

Definition 6.10. *An extrapolation* ∇ *is* restricted *if for every* $k \in \mathbb{N}$*, the set* $\{\nabla_k(L) \mid L \in \mathcal{R}ec((M^*)^n)\}$ *is finite.*

Remark that the extrapolation used in Example 6.9 was not restricted. The path invariants obtained in this example only used ∇_0, as the **while**-loop of the algorithm UPInv was never repeated. The use of restricted extrapolations prevents this kind of executions. Indeed, as a consequence of Proposition 6.2, we obtain that if ∇ is restricted then for any execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad,$

$P_0, \mathsf{UPInv}(\nabla)$), the execution terminates if and only if the number of iterations of the **while**-loop of the algorithm UPInv is bounded.(Remark that this bound is not a bound on the length of abstract counterexamples.) As shown by the following proposition, if moreover $[\![\mathcal{A}]\!]$ has a finite reachability set from $Init$ then the execution necessarily terminates.

Proposition 6.11. *Assume that ∇ is restricted. If $[\![\mathcal{A}]\!]$ has a finite reachability set from $Init$, then any execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$ terminates.*

Proof. Assume that $[\![\mathcal{A}]\!]$ has a finite reachability set from $Init$, and consider any execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$. For each $q \in Q$, let us write $RS(q)$ the finite set of $\boldsymbol{w} \in (M^*)^n$ such that there is a path in $[\![\mathcal{A}]\!]$ from $Init$ to $(q, \boldsymbol{w})$. Define $\mathcal{L} = \bigcup_{q \in Q} RS(q)$ and remark that $\mathcal{L}$ is finite. Recall that according to Definition 4.3, for any recognizable subset L of $(M^*)^n$, there exists $k_L \in \mathbb{N}$ such that $L = \nabla_k(L)$ for every $k \geq k_L$. Since $\mathcal{L}$ is finite, we infer that there exists $K \in \mathbb{N}$ such that $L = \nabla_k(L)$ for every $k \geq K$ and $L \subseteq \mathcal{L}$. Let us define $\mathcal{H} = \wp(\mathcal{L}) \cup \{\nabla_k(L) \mid k < K, L \in \mathcal{R}ec((M^*)^n)\}$. Observe that $\mathcal{H}$ is finite since ∇ is restricted.

We show that $L_i^\theta \in \mathcal{H}$ for any iteration θ and for any $0 \leq i \leq h_\theta$. Consider an iteration θ, and let us write $\pi_\theta^\sharp$ as $\pi_\theta^\sharp = (q_0, p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, p_h)$, with $h = h_\theta$. Notice that (q_{i-1}, l_{i-1}, q_i) is a transition rule in Δ for each $1 \leq i \leq h$. Let us define $R_0 = p_0 \cap \{\boldsymbol{w} \mid (q_0, \boldsymbol{w}) \in Init\}$ and $R_i = post(l_{i-1}, p_{i-1} \cap L_{i-1}^\theta)$ for every $1 \leq i \leq h$. We derive from the definition of the algorithm UPInv that there exists $k \in \mathbb{N}$ such that: $L_0^\theta = \nabla_k(R_0)$, and $L_i = \emptyset$ or $L_i = \nabla_k(R_i)$ for every $1 \leq i \leq h$. If $k < K$ then we get that $L_i^\theta \in \mathcal{H}$ for every $0 \leq i \leq h$. Otherwise, we have $k \geq K$ and therefore $L_i = R_i$ for every $0 \leq i \leq h$. An immediate induction on i shows that $R_i \subseteq RS(q_i)$ for every $0 \leq i \leq h$. We deduce that $L_i^\theta \subseteq \mathcal{L}$ and hence $L_i^\theta \in \mathcal{H}$ for every $0 \leq i \leq h$.

We obtain that $\left\{L_i^\theta \;\middle|\; \theta \in \mathbb{N} \text{ and } 0 \leq i \leq h_\theta\right\} \subseteq \mathcal{H}$. Since $\mathcal{H}$ is finite, we conclude with Proposition 6.2 that the execution terminates. □

Remark 6.12. Our notion of restricted extrapolation is related to Jhala and McMillan's *restricted interpolation*, introduced in [JM06] to derive partial completeness results of CEGAR for predicate abstraction. Indeed, given a restricted extrapolation ∇, the finite subsets $L_k = \{\nabla_i(L) \mid i \leq k, L \in \mathcal{R}ec((M^*)^n)\}$ of $\mathcal{R}ec((M^*)^n)$ are analogous to the finite languages L_k of predicates that restrict interpolants in [JM06]. However, in contrast to the CEGAR semi-algorithm of [JM06] where the restricted languages of predicates are enlarged globally, the parameter k is tailored to each abstract counterexample in our CEGAR semi-algorithm combined with $\mathsf{UPInv}(\nabla)$ or $\mathsf{APInv}(\nabla)$.

7 Experimental Evaluation

We implemented our ideas in the framework McScM that includes prototypical tools implementing our previous ideas. McScM is written in OCaml and relies

on a library by Le Gall and Jeannet [SCM] for the classical finite automata and QDD operations, the fifo *post*/*pre* symbolic computations, as well as the colored bisimulation-based extrapolation. The standard coloring with final and non-final states is used by default in our tool (see Section 4.3), but several other variants are available in addition.

We implemented the CEGAR semi-algorithm of Section 5 as a generic OCAML functor, that is parameterized by a symbolic representation of the model to be verified. This should allow us to transfer the ideas of this paper to other classes of infinite-state models such as counter automata, hybrid automata, etc. The initial partition map of the CEGAR semi-algorithm is fixed to the trivial one $\lambda q\,.\,(M^*)^n$, hence, the initial partition abstraction is "isomorphic" to the "control flow graph" of the input fifo system.

We tested the prototype on a suite of protocols that includes the classical alternating bit protocol (ABP) [AJ96], a simplified version of TCP—also in the setting of one server with two clients that interfere on their shared channels, a sliding window protocol, as well as protocols for leader election due to Peterson and token passing in a ring topology. Further, we provide certain touchstones for our approach: an enhancement of the c/d protocol with nested loops for the exchange of data, and a protocol with a non-recognizable reachability set. A detailed presentation of the protocols is provided in Appendix C. Except for the c/d protocol, which is unsafe, all other examples are safe.

To increase confidence in the results, an independent module, that can perform inductive invariant checking and feasibility checking, verifies the result of the CEGAR implementation.

All following results were obtained by our checker implemented in the MCSCM framework (version 1.0, [McScM]) on an off-the-shelf computer (3.2 GHz Intel i7-965, 64-bit system).

7.1 Benchmark

Section 4 introduces the two path invariant generation algorithms UPInv and APInv (both in "forward" direction, abbreviated ▶I in the following) based on forward-bisimulation-based extrapolation (Definition 4.3). As already mentioned, we can also reverse both invariant generation algorithms (noted I◀). In the following, we compare the four invariant generation algorithms and additionally include backward-bisimulation-based extrapolation. Figure 10 shows for each of our protocols the best and worst combination with respect to running time. The first columns introduce the protocol, the size of its control structure, and whether the protocol is bounded (denoted "b") or not (ω). The last rows give the number of refinement loops and the size of the final abstract transition system.

When comparing the different path invariant generation procedures of Figure 10, we are not able to deduce a favorable method; for example, the *adaptive* algorithm in its I◀ variant together with *forward* bisimulation-based extrapolation seems encouraging after the first examples, whereas it does not lead to a

protocol	states/trans.	B	pinv-gen/bisim	time [s]	mem [MiB]	loops	states$^\sharp$/trans$^\sharp$
ABP	16/64	ω	APInv ⏮/ fwd	0.25	3.94	47	90/528
			APInv ⏮/ bwd	1.28	5.88	220	301/1931
c/d	5/17	ω	APInv ⏮/ fwd	0.00	2.97	6	11/34
			UPInv ⏮/ bwd	0.02	2.97	6	12/47
nested c/d	6/17	ω	APInv ⏮/ fwd	0.43	3.94	65	81/313
			APInv ⏭/ fwd	13.85	16.53	244	261/1173
non-regular	9/18	b	APInv ⏮/ fwd	0.01	2.97	8	21/47
			UPInv ⏭/ bwd	0.03	2.97	19	31/39
Peterson	10k/57k	b	APInv ⏭/ bwd	0.53	38.81	33	10689/56802
			UPInv ⏭/ fwd	499.52	251.94	10641	25548/206681
(simplified) TCP	196/588	ω	APInv ⏭/ fwd	0.25	3.94	98	323/987
			UPInv ⏮/ fwd	($> 1h$)	—	—	—
server/2 clients	255/2160	ω	UPInv ⏭/ bwd	4.81	9.75	398	714/7192
			APInv ⏮/ fwd	($> 1h$)	—	—	—
token ring	625/4500	b	APInv ⏭/ fwd	0.49	9.75	132	760/5267
			UPInv ⏮/ bwd	79.79	109.53	14527	18640/88598
sliding window	225/2010	b	APInv ⏭/ fwd	0.50	5.88	162	399/2420
			UPInv ⏮/ bwd	48.54	39.78	3727	5113/19650
BRP	100/510	ω	UPInv ⏭/ fwd	1.59	6.84	149	310/1892
			APInv ⏮/ bwd	($> 1h$)	—	—	—
POP3	460/2018	b	UPInv ⏭/ bwd	0.38	5.88	86	564/2245
			APInv ⏭/ fwd	($> 1h$)	—	—	—

Fig. 10: Benchmark Comparing Path Invariant Generation Algorithms (Excerpt)

result after $1h$ in the case of the extended TCP protocol where two clients interfere. Analogous considerations hold for all other combinations. However, our approach was able to allow safety verfication for our set of example protocols using relatively few ressources (time and memory).

Comparison to ARMC

Notice that in ARMC, abstraction is performed on the data structures that are used to represent sets of configurations, whereas in our case the system itself is abstracted. After each refinement step, ARMC restarts (from scratch) the approximated forward exploration from the refined reachability set, whereas our refinement is *local* to the spurious counterexample path. Moreover, the precision of the abstraction is *global* in ARMC, and may only increase (for the entire system) at each refinement step. In contrast, our path invariant generation procedures only uses the precision *required* to rule out each found spurious counterexample. Preliminary benchmarks demonstrate the benefit of our local and adaptive approach for the larger examples, where a "highly" precise abstraction is required only for a small part of the model and the remaining model can be coarsely abstracted. Last, our approach is not tied to words and automata. In this work we only focus on fifo systems, but our framework is fully generic and could be applied to other infinite-state systems (e.g., hybrid systems), provided that suitable parametrized extrapolations are designed (e.g., on polyhedra).

Rerunning the previous benchmark on our own re-implementation of ARMC suggests that the latter seems to be advantageous for some small protocols. However, larger examples confirm that our local and adaptive refinement approach outperforms a global refinement one in protocols that demand a "highly" precise abstraction only for a few control loops (e.g., Peterson's leader election and token ring). Further, our ARMC reimplementation was not able to treat the non-regular protocol nor the case of the server with 2 clients.

Comparison to ABSINT
An orthogonal approach to ours is abstract interpretation which was implemented for fifo systems in [LGJJ06]. This publication is the origin of the previously applied bisimulation-based extrapolation which was introduced as widening. The proposed ABSINT approach is also a semi-algorithm, since we increse the parameter of the widening until the sytem is proven to be safe. Obviously, this semi-algorithm cannot terminate if the system is unsafe and does not lead to counterexamples. Our tool provides an interface to ABSINT that is able to verify all safe protocols in seconds. Some additional experiments where we introduced errors in the originally safe protocols confirmed that ABSINT is an ideal preprocessing step to test whether the protocol is safe, but searching for counterexamples should be done with our CEGAR approach.

Comparison to Other Tools

We compared our tool with TREX, which is, to the best of our knowledge, the sole publicly available (see [TReX]) and directly usable model-checker for the verification of *unbounded* fifo systems. Note, however, that the comparison is biased as TREX focuses on *lossy* channels. We applied TREX to the first six protocols of our benchmark (see Figure 10). TREX has an efficient implementation based on *simple regular expressions* (and not general QDDs as we do), and needs for our examples less than 1 second to build the reachability set for the protocols in the upper part of the previous table. (Given the reachability set, deciding the reachability of bad configurations becomes a simple additional look-up.) Further, TREX implements communicating timed and counter automata that are—at this stage—beyond the focus of our framework. Nonetheless, TREX assumes a *lossy* fifo semantics, and, therefore, is not able to verify all *reliable* fifo examples correctly (e.g., when omitting the `disconnect` messages in the c/d protocol, TREX is still able to reach *Bad* due to the possible loss of messages, albeit the protocol is safe).

Our tool also allows to handle lossy channels. However, it does not include an optimized symbolic representation like TREX. Yet one can easily mix lossy and reliable channels in the same model.

Moreover, TREX suffers (as would also a symbolic model checker based on the LASH library [Lash]) from the main drawback of acceleration techniques, which in general cannot cope with nested loops, whereas these loops seem to have no adverse effect on our tool (viz. nested c/d protocol on which TREX did not

finish after one hour). Our tool can also handle a simple non-regular protocol (with a counting loop) that is beyond the QDD-based approaches [BG99], as the representation of the reachability set would require recognizable languages equipped with Presburger formulas (i.e., CQDDs [BH99]).

The highly optimized model checker SPIN [SPIN] is an alternative due to its importance in practice. Even tough it is based on an underlying semantic model of communicating automata (with global variables, dynamic process generation,...), it is strongly limited as it demands *bounded* channels. Engineering folklore states that most errors can already be found within a small bound on the size of the channels, hence SPIN is useful due to its thorough search of the resulting finite transition system for counterexamples. On the other hand, SPIN can only give the guarantee that the system is safe under the additional assumption that the channels are a priori bounded and cannot prove general safety for infinite state systems as demanded by our basic domain. Nonetheless, the memory consumption of SPIN rises drastically, if one goes beyond small channel bounds, even for trivial protocols like the previous ones. For example, running the nested version of the c/d protocol with a channel bound of 30 leads to a model of 9227463 states and 1854922 transitions that can exhaustively be searched in $19s$ but with a memory consumption of around $1.4GB$ (SPIN 6.0.1 on same machine as above). An interesting approach is to marry SPIN with symbolic techniques or abstraction [dMGMMP02] which, however, never gained wider attention. Then again, also our approach could benefit from partial order exploration and state-space reduction techniques as applied in SPIN.

A different approach relies on the application of machine learning techniques to state space exploration [VSVA04a, VSVA04b]. The basic idea is to apply Angluin's L^* algorithm [Ang87] to generate a deterministic finite automata representing an abstraction of the set of reachable configurations (by *regular* traces) which is then combined with a classical abstract-check-refine algorithm. The previously cited articles were accompanied by a prototypical implementation in the LEVER tool. Regrettably, the current public version does not further support fifo automata but only parametrized systems that communicate over shared variables (over a finite domain), such that no direct comparison is possible.

8 Conclusion and Perspectives

Our prototypical implementation confirms our expectations that the proposed CEGAR framework with extrapolation-based path invariants is a promising alternative approach to the automatic verification of fifo systems.

Our approach relies on partition abstractions where equivalence classes are recognizable languages of queue contents. Our main contribution is the design of generic path invariant generation algorithms based on parameterized extrapolation operators for queue contents. Because of the latter, our CEGAR semi-algorithm enjoys additional partial termination properties.

The framework developed in this paper is not specific to fifo systems, and we intend to investigate its practical relevance to other families of infinite-state models. Future work also includes the safety verification of more complex fifo systems that would allow the exchange of unbounded numerical data over the queues, or include parameterization (e.g., over the number of clients). Several decidable classes of fifo systems have emerged in the literature (in particular lossy fifo systems) and we intend to investigate termination of our CEGAR semi-algorithm (when equipped with the path invariant generation algorithms developed in this paper) for these classes. A fully automatic approach to safety verification would further demand an appropriate heuristics with respect to different classes of communication protocols, in order to choose the appropriate path invariant generation algorithm, graph search method, etc.

References

[AJ96] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. 127(2):91–101, 1996.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. 75(2):87–106, 1987.

[Ans08] Amin Ansari. Verification of Peterson's algorithm for leader election in a unidirectional asynchronous ring using NuSMV. *Computing Research Repository*, abs/0808.0962, 2008.

[Ber79] Jean Berstel. *Transductions and Context-Free Languages*. Teubner, 1979.

[BG99] Bernard Boigelot and Patrice Godefroid. Symbolic verification of communication protocols with infinite state spaces using qdds. 14(3):237–255, 1999.

[BGWW97] Bernard Boigelot, Patrice Godefroid, Bernard Willems, and Pierre Wolper. The Power of QDDs. In *Proc. of SAS'97*, LNCS 1302, pages 172–186, 1997.

[BH99] Ahmed Bouajjani and Peter Habermehl. Symbolic Reachability Analysis of FIFO-Channel Systems with Nonregular Sets of Configurations. *Theoretical Computer Science*, 221(1-2):211–250, 1999.

[BHV04] Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Abstract regular model checking. In *Proc. of CAV'04*, LNCS 3114, pages 372–386, 2004.

[BR01] Thomas Ball and Sriram K. Rajamani. Automatically validating temporal safety properties of interfaces. In *Proc. of SPIN'01*, LNCS 2057, pages 103–122, 2001.

[BZ83] Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *Journal of the ACM*, 30(2):323–342, 1983.

[CGJ+03] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided Abstraction Refinement for Symbolic Model Checking. *Journal of the ACM*, 50(5):752–794, 2003.

[dMGMMP02] María del Mar Gallardo, Jesús Martínez, Pedro Merino, and Ernesto Pimentel. alpha SPIN: Extending SPIN with abstraction. In *Proc. of SPIN'02*, number 2318, pages 254–258, 2002.

[FIS03] A. Finkel, S. Purushothaman Iyer, and G. Sutre. Well-Abstracted Transition Systems: Application to FIFO Automata. 181(1):1–31, 2003.

[GS97] Susanne Graf and Hassen Saïdi. Construction of Abstract State Graphs with PVS. In *Proc. of CAV'97*, LNCS 1245, pages 72–83, 1997.

[HJMS02] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. Lazy abstraction. In *SIGPLAN-SIGACT Symposium on Principles of Programming Languages 2002*, pages 58–70, 2002.

[HLS09] Alexander Heußner, Tristan Le Gall, and Grégoire Sutre. Extrapolation-based path invariants for abstraction refinement of fifo systems. In *Proc. of SPIN'09*, LNCS 5578, pages 107–124, 2009.

[JM06] Ranjit Jhala and Kenneth L. McMillan. A practical and complete approach to predicate refinement. In *Proc. of TACAS'06*, number 3920, pages 459–473, 2006.

[JR86] Claude Jard and Michel Raynal. De la nécessité de spécifier des propriétés pour la verification des algorithmes distribués. Rapports de Recherche 590, IRISA Rennes, December 1986.

[Lash] Liège Automata-based Symbolic Handler Toolset (Lash). Tool Homepage. `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`

[Lever] Tool for Learning based Verification (Lever). Tool Homepage. `http://abhayspace.com/static/lever.html`

[LGJJ06] T. Le Gall, B. Jeannet, and T. Jéron. Verification of Communication Protocols using Abstract Interpretation of FIFO queues. In *Proc. of AMAST'06*, LNCS 4019, pages 204–219, 2006.

[McScM] Model Checker for Systems of Communicating Fifo Machines (McScm). Tool Homepage. `http://altarica.labri.fr/forge/projects/mcscm/wiki`

[Pet82] Gary L. Peterson. An o(n log n) unidirectional algorithm for the circular extrema problem. *ACM Trans. Program. Lang. Syst.*, 4(4):758–762, 1982.

[SCM] Tools and Libraries for Static Analysis and Verification. Tool Homepage. `http://gforge.inria.fr/projects/bjeannet/`.

[SPIN] Simple Promela Interpreter. Tool Homepage. `http://www.SPINroot.com`

[TReX] Tool for Reachability Analysis of CompleX Systems. Tool Homepage. `http://www.liafa.jussieu.fr/~sighirea/trex/`

[VSVA04a] Abhay Vardhan, Koushik Sen, Mahesh Viswanathan, and Gul Agha. Actively learning to verify safety for fifo automata. In *Proc. of FSTTCS'04*, LNCS 3323, pages 494–505, 2004.

[VSVA04b] Abhay Vardhan, Koushik Sen, Mahesh Viswanathan, and Gul Agha. Learning to verify safety properties. In *Proc. of ICFEM'04*, number 3308, pages 274–289, 2004.

[YBCI08] Fang Yu, Tevfik Bultan, Marco Cova, and Oscar H. Ibarra. Symbolic string verification: An automata-based approach. In *Proc. of SPIN'08*, LNCS 5156, pages 306–324, 2008.

Appendix

A Queue Decision Diagrams

Boigelot and Godefroid introduced queue decision diagrams (QDDs) for the symbolic verification of infinite-state communication protocols [BG99], and an implementation of QDDs is provided in the LASH toolset [Lash]. We recall in this section the definition and main properties of QDDs. We adapt the original definition of [BG99, BGWW97], which assumed disjoint queue alphabets, to our framework where a single alphabet is used for all queues. Moreover, to simplify the presentation of extrapolations in the next section, we will w.l.o.g. restrict our attention to *trim* QDDs, which have no useless transitions. This restriction is important to determine easily the *dimension* of a QDD, i.e. the number of queue content it represents.

A *finite automaton* is any 5-tuple $\mathcal{D} = \langle Q, \Sigma, \rightarrow, I, F\rangle$ where $\langle Q, \Sigma, \rightarrow\rangle$ is a finite labeled transition system whose configurations are called *states*, $I \subseteq Q$ is a set of *initial states*, and $F \subseteq Q$ is a set of *final states*. A word $w = l_0 \cdots l_{h-1} \in \Sigma^*$ is *accepted* by $\mathcal{D}$ if there is a run $q_0 \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} q_h$ such that $q_0 \in I$ and $q_h \in F$. The *accepted language* of a finite automaton $\mathcal{D}$, written $L(\mathcal{D})$, is the set of all words accepted by $\mathcal{D}$. We say that $\mathcal{D}$ is *trim* when every state $q \in Q$ occurs on some accepting run (from I to F). Remark that if $\mathcal{D}$ is trim then: $L(\mathcal{D}) = \emptyset$ if and only if $\mathcal{D}$ has an empty set of states. In the remainder of this paper, we assume that all finite automata in the following are always trim.

For any letter $a \in \Sigma$ and word $w \in \Sigma^*$, we let $|w|_a$ denote the number of occurences of a in w.

Consider a finite alphabet M of messages and an integer $n \geq 1$ denoting the number of queues. We use the classical encoding of n-tuples $\boldsymbol{w} \in (M^*)^n$ by words over $M \cup \{\mathbb{\in}\}$ where $\mathbb{\in} \notin M$ is a new letter used as *separator*. Formally, we define the function $\eta : (M^*)^n \rightarrow (M \cup \{\mathbb{\in}\})^*$ by $\eta(\boldsymbol{w}) = \boldsymbol{w}(1)\mathbb{\in}\cdots\mathbb{\in}\boldsymbol{w}(n)$. Let us write $E(M, n)$ for the set of all words $w \in (M \cup \{\mathbb{\in}\})^*$ with $|w|_{\mathbb{\in}} = n-1$. Notice that η is a bijection between $(M^*)^n$ and $E(M, n)$.

Definition A.1. *An n-dim* queue decision diagram *for M is any (trim) finite automaton $\mathcal{D} = \langle Q, M \cup \{\mathbb{\in}\}, \rightarrow, I, F\rangle$ such that $L(\mathcal{D}) \subseteq E(M, n)$.*

We denote by $\mathcal{Q}dd\,(M, n)$ the set of all n-dim queue decision diagrams for M. For notational convenience, we write $[\![\mathcal{D}]\!]$ for the subset of $(M^*)^n$ represented by a queue decision diagram $\mathcal{D} \in \mathcal{Q}dd\,(M, n)$, defined by $[\![\mathcal{D}]\!] = \eta^{-1}(L(\mathcal{D}))$. The subsets of $(M^*)^n$ that are representable by queue decision diagrams coincide exactly with the recognizable subsets, formally:

Theorem A.2 ([BGWW97]). *For any finite alphabet M and integer $n \geq 1$, the following equality holds:*

$$\mathcal{R}ec\,((M^*)^n) \;=\; \{[\![\mathcal{D}]\!] \mid \mathcal{D} \in \mathcal{Q}dd\,(M, n)\}$$

B Quotient-based Extrapolations

In this section, we show how well-known behavioral equivalences on labeled transition systems can be turned into extrapolations (that satisfy the requirements of Definition 4.3).

Consider a finite alphabet M and an integer $n \geq 1$. Recall that recognizable subsets of $(M^*)^n$ are in bijection with languages accepted by QDDs. We assume for the remainder of this section a function χ from $\mathcal{R}ec\,((M^*)^n)$ to $\mathcal{Q}dd\,(M,n)$ such that $L = [\![\chi(L)]\!]$ for every $L \in \mathcal{R}ec\,((M^*)^n)$. With the help of the function η of Section A, this function χ could be defined for instance by mapping each recognizable subset L of $(M^*)^n$ to the minimal deterministic finite automaton accepting $\eta(L)$. Thanks to the function χ, the design of extrapolations for recognizable subsets can be reduced to the design of extrapolations for QDDs.

Definition B.1. *A* (parametrized) QDD-extrapolation *is any function* ∇ *from* $\mathbb{N}$ *to* $\mathcal{Q}dd\,(M,n) \to \mathcal{Q}dd\,(M,n)$ *that satisfies, for any* $\mathcal{D} \in \mathcal{Q}dd\,(M,n)$*, the two following conditions (with* $\nabla(k)$ *written as* ∇_k*):*

(i) we have $L(\mathcal{D}) \subseteq L(\nabla_k(\mathcal{D}))$ *for every* $k \in \mathbb{N}$*,*
(ii) there exists $k_{\mathcal{D}} \in \mathbb{N}$ *such that* $L(\mathcal{D}) = L(\nabla_k(\mathcal{D}))$ *for every* $k \geq k_{\mathcal{D}}$*.*

Definition B.2. *A* QDD*-extrapolation* ∇ *is* restricted *if for every* $k \in \mathbb{N}$*, the set* $\{\nabla_k(\mathcal{D}) \mid \mathcal{D} \in \mathcal{Q}dd\,(M,n)\}$ *is finite up to automata isomorphism.*

For any QDD-extrapolation ∇, the function $\nabla^\chi = \lambda k.(\lambda L.[\![\nabla_k(\chi(L))]\!])$ is an extrapolation (in the sense of Definition 4.3). Moreover, if ∇ is restricted then ∇^χ is restricted (in the sense of Definition 6.10). We thus focus on the design of (restricted) QDD-extrapolations for the remainder of this section.

The *quotient* $\mathcal{D}/{\equiv}$ of a finite automaton $\mathcal{D} = \langle Q, \Sigma, \to, I, F\rangle$ by an equivalence relation $\equiv$ on Q is the finite automaton $\langle Q^{\equiv}, \Sigma, \to^{\equiv}, I^{\equiv}, F^{\equiv}\rangle$ where:

$$\begin{array}{rclcrcl} Q^{\equiv} & = & Q/{\equiv} & & I^{\equiv} & = & \{[\,q\,]_{\equiv} \mid q \in I\} \\ \to^{\equiv} & = & \left\{([\,q\,]_{\equiv}, l, [\,q'\,]_{\equiv}) \;\middle|\; q \xrightarrow{l} q'\right\} & & F^{\equiv} & = & \{[\,q\,]_{\equiv} \mid q \in F\} \end{array}$$

With this definition, $\mathcal{D}/{\equiv}$ is trim whenever $\mathcal{D}$ is trim. Moreover, it holds that $L(\mathcal{D}) \subseteq L(\mathcal{D}/{\equiv_1}) \subseteq L(\mathcal{D}/{\equiv_2})$ for any two equivalence relations $\equiv_1$ and $\equiv_2$ on Q such that $\equiv_1 \subseteq \equiv_2$.

Given an n-dim queue decision diagram $\mathcal{D} = \langle Q, M \cup \{\in\!\!\!\!\!\in\}, \to, I, F\rangle$ for M; we define a set $Q_i \subseteq Q$ for $1 \leq i \leq n$: $q \in Q$ if there exist $q_0 \in I$ and $q_h \in F$ such that there exists a path $q_0 \xrightarrow{l_0} \ldots \xrightarrow{l_{i-1}} q_i \xrightarrow{l_i} \ldots \xrightarrow{l_{h-1}} q_h$ with $l_i \in M \cup \{\in\!\!\!\!\!\in\}$ for $0 \leq i < h$ such that $|l_0 \cdots l_{i-1}|_{\in\!\!\!\!\!\in} = i - 1$ and $|l_i \cdots l_{h-1}|_{\in\!\!\!\!\!\in} = n - i$. Clearly, if $Q \neq \emptyset$ then $\{Q_i\}_{i=1}^n$ is a partition of Q since $\mathcal{D}$ is trim.

We define the equivalence relation $\approx_{\mathcal{D}}$ on Q as follows: if $Q = \emptyset$ then $\approx_{\mathcal{D}} = \emptyset$, otherwise $\approx_{\mathcal{D}} = \bigcup_{i=1}^n Q_i \times Q_i$. The following two propositions show that $\approx_{\mathcal{D}}$ is the coarsest equivalence relation on Q under which the quotient of $\mathcal{D}$ is also a QDD.

Proposition B.3. *For any n-dim queue decision diagram $\mathcal{D}$ for M, the quotient $\mathcal{D}/{\approx_{\mathcal{D}}}$ is an n-dim queue decision diagram for M. Moreover, if $[\![\mathcal{D}]\!] \neq \emptyset$ then $[\![\mathcal{D}/{\approx_{\mathcal{D}}}]\!] = N_1^* \times \cdots \times N_n^*$ where $N_i = alph\,(\{\boldsymbol{w}(i) \mid \boldsymbol{w} \in [\![\mathcal{D}]\!]\})$ for all $1 \leq i \leq n$.*

Proof. Let us write $\mathcal{D} = \langle Q, M \cup \{\Subset\}, \rightarrow, I, F\rangle$. If Q is empty then $\mathcal{D}/{\approx_{\mathcal{D}}} = \mathcal{D}$ and hence $\mathcal{D}/{\approx_{\mathcal{D}}}$ is also in $\mathcal{Qdd}\,(M,n)$. Assume now that Q is non-empty. We first give an explicit characterization of $\mathcal{D}/{\approx_{\mathcal{D}}}$. Observe that $I \subseteq Q_1$ and $F \subseteq Q_n$.

Consider any two integers i, j in $\{1, \ldots, n\}$ and assume that there is a transition $r \xrightarrow{l} r'$ in $\mathcal{D}$ with $r \in Q_i$ and $r' \in Q_j$. Since $\mathcal{D}$ is trim, there exists in $\mathcal{D}$ two paths $q \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} r$ and $r' \xrightarrow{l'_0} \cdots \xrightarrow{l'_{h'-1}} q'$ such that $q \in I$ and $q' \in F$. Note that $|l_0 \cdots l_{h-1}|_{\Subset} = i-1$ since $r \in Q_i$. Two cases arise:

- if $l = \bowtie$ then we obtain that $r' \in Q_{i+1}$ since $q \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} r \xrightarrow{\Subset} r'$ is a path in $\mathcal{D}$ and $|l_0 \cdots l_{h-1} \Subset |_{\Subset} = i$. We deduce that $j = i+1$.
- if $l \in M$ then we obtain that $r' \in Q_i$ since $q \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} r \xrightarrow{l} r'$ is a path in $\mathcal{D}$ and $|l_0 \cdots l_{h-1} l|_{\Subset} = i-1$. We deduce that $j = i$. Moreover, we have $l \in N_i$ since the n-tuple $\boldsymbol{w} = \eta^{-1}(l_0 \cdots l_{h-1} \cdot l \cdot l'_0 \cdots l'_{h'-1})$ satisfies $\boldsymbol{w} \in [\![\mathcal{D}]\!]$ and $\boldsymbol{w}(i) \in M^* \cdot l \cdot M^*$.

Conversely, since $[\![D]\!] \neq \emptyset$, there exists $w \in L(\mathcal{D})$. We may write w as $w = l_0^1 \cdots l_{h^1-1}^1 \Subset \cdots \Subset l_0^n \cdots l_{h^n-1}^n$ with $l_0^i \cdots l_{h^i-1}^i \in M^*$ for all $1 \leq i \leq n$. Therefore, there exists in $\mathcal{D}$ a path $q_0^1 \xrightarrow{l_0^1} \cdots \xrightarrow{l_{h^1-1}^1} q_{h^1}^1 \xrightarrow{\Subset} \cdots \xrightarrow{\Subset} q_0^n \xrightarrow{l_0^n} \cdots \xrightarrow{l_{h^n-1}^n} q_{h^n}^n$ with $q_0^1 \in I$ and $q_{h^n}^n \in F$. We get that q_0^i and $q_{h^i}^i$ are in Q_i for all $1 \leq i \leq n$, and moreover $q_{h^i}^i \xrightarrow{\Subset} q_0^{i+1}$ for all $1 \leq i < n$. We deduce that for every $1 \leq i < n$, there exists in $\mathcal{D}$ a transition $q \xrightarrow{\Subset} q'$ with $q \in Q_i$ and $q' \in Q_{i+1}$.

Consider now any $i \in \{1, \ldots, n\}$ and let $l \in N_i$. There exists $\boldsymbol{w} \in [\![\mathcal{D}]\!]$ such that $\boldsymbol{w}(i) \in M^* \cdot l \cdot M^*$. Since $\eta(\boldsymbol{w}) = \boldsymbol{w}(1) \Subset \cdots \Subset \boldsymbol{w}(n) \in L(\mathcal{D})$, there exists in $\mathcal{D}$ a path $q_0 \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} q_h \xrightarrow{l} q_{h+1}$ with $q_0 \in I$ and $|l_0 \cdots l_{h-1}|_{\Subset} = i-1$. We deduce that there exists in $\mathcal{D}$ a transition $q \xrightarrow{l} q'$ with $q, q' \in Q_i$.

We have thus shown that the quotient $\mathcal{D}/{\approx_{\mathcal{D}}}$ is the trim finite automaton $\langle Q^{\approx}, \Sigma, \rightarrow^{\approx}, I^{\approx}, F^{\approx}\rangle$ where:

$$\begin{array}{rcl} Q^{\approx} & = & \{Q_i \mid 1 \leq i \leq n\} \\ \rightarrow^{\approx} & = & \{(Q_i, l, Q_i) \mid 1 \leq i \leq n, l \in N_i\} \cup \{(Q_i, \Subset, Q_{i+1}) \mid 1 \leq i < n\} \\ I^{\approx} & = & \{Q_1\} \\ F^{\approx} & = & \{Q_n\} \end{array}$$

We derive that $L(\mathcal{D}/{\approx_{\mathcal{D}}}) = N_1^* \Subset \cdots \Subset N_n^* \subseteq E(M,n)$, which entails that $\mathcal{D}/{\approx_{\mathcal{D}}}$ is an n-dim queue decision diagram for M. Moreover, we also get that $[\![\mathcal{D}/{\approx_{\mathcal{D}}}]\!] = \eta^{-1}(N_1^* \Subset \cdots \Subset N_n^*) = N_1^* \times \cdots \times N_n^*$. □

It follows from the previous proposition that for any equivalence relation $\equiv \subseteq \approx_{\mathcal{D}}$, the quotient $\mathcal{D}/{\equiv}$ is also an n-dim queue decision diagram for M. The following proposition shows that the converse also holds.

Proposition B.4. *Let $\mathcal{D} = \langle Q, M \cup \{\in\}, \to, I, F\rangle$ be an n-dim queue decision diagram for M. For any equivalence relation $\equiv$ on Q, if the quotient $\mathcal{D}/{\equiv}$ is an n-dim queue decision diagram for M then it holds that $\equiv \subseteq \approx_{\mathcal{D}}$.*

Proof. Assume that $\equiv \not\subseteq \approx_{\mathcal{D}}$ and let us prove that $\mathcal{D}/{\equiv} = \langle Q^{\equiv}, \Sigma, \to^{\equiv}, I^{\equiv}, F^{\equiv}\rangle$ is not an n-dim queue decision diagram for M. Note that $Q \neq \emptyset$ since otherwise $\equiv$ and $\approx_{\mathcal{D}}$ would be empty. Moreover, we deduce from $\equiv \not\subseteq \approx_{\mathcal{D}}$ that there exists $1 \leq i < j \leq n$ and $(r, r') \in Q_i \times Q_j$ such that $r \equiv r'$. Since $\mathcal{D}$ is trim, there exists in $\mathcal{D}$ two paths $q \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} r$ and $r' \xrightarrow{l'_0} \cdots \xrightarrow{l'_{h'-1}} q'$ such that $q \in I$ and $q' \in F$. Therefore, $[\,q\,]_{\equiv} \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} [\,r\,]_{\equiv} = [\,r'\,]_{\equiv} \xrightarrow{l'_0} \cdots \xrightarrow{l'_{h'-1}} [\,q'\,]_{\equiv}$ is a path in $\mathcal{D}/{\equiv}$ from $I^{\equiv}$ to $F^{\equiv}$. Hence the word $w = l_0 \cdots l_{h-1} l'_0 \cdots l'_{h'-1}$ is accepted by $\mathcal{D}/{\equiv}$. Since $r \in Q_i$ and $r' \in Q_j$, we get that $|w|_{\in} = (i-1) + (n-j)$. As $i \neq j$, we conclude that $w \notin E(M, n)$ and hence $L(\mathcal{D}/{\equiv}) \not\subseteq E(M, n)$, which entails that $\mathcal{D}/{\equiv}$ is not in $\mathcal{Q}dd\,(M, n)$. □

Regarding the algorithms UPInv and APInv of Section 4, we focus on quotient-based QDD-extrapolations that lead to suitable path invariant generation algorithms for fifo systems. Natural candidates are bounded-depth behavioral equivalences such as bounded languages equivalence and bisimulation equivalence. The former was used in [BHV04] to over-approximate finite automata in abstract regular model checking. The latter was used in [LGJJ06] to derive a widening operator in abstract interpretation of fifo systems with QDDs.

B.1 Colored Bisimulation-based Extrapolation

We recall in this subsection the extrapolation underlying the widening operator introduced in [LGJJ06]. This extrapolation relies on bounded-depth bisimulation based on an initial coloring that partitions the set of states. The extrapolation presented in [LGJJ06] relied on minimal deterministic automata. Requiring minimization at each extrapolation step may adversely affect performance in practice. We extend in this subsection the approach of [LGJJ06] to arbitrary automata.

Definition B.5. *Let $\mathcal{D} = \langle Q, \Sigma, \to, I, F\rangle$ be a finite automaton, and let col be an equivalence relation on Q. For every $k \in \mathbb{N}$, the* bisimulation equivalence of depth k *is the relation $\sim_k^{col}$ on Q defined inductively by:*

$$q_1 \sim_0^{col} q_2 \quad \textit{if} \quad (q_1, q_2) \in col$$

$$q_1 \sim_{k+1}^{col} q_2 \quad \textit{if} \quad \begin{cases} q_1 \sim_k^{col} q_2 \\ \forall l \in \Sigma, \forall q'_1 \in Q : q_1 \xrightarrow{l} q'_1 \Rightarrow \left(\exists q'_2 \in Q : q'_1 \sim_k^{col} q'_2 \wedge q_2 \xrightarrow{l} q'_2\right) \\ \forall l \in \Sigma, \forall q'_2 \in Q : q_2 \xrightarrow{l} q'_2 \Rightarrow \left(\exists q'_1 \in Q : q'_1 \sim_k^{col} q'_2 \wedge q_1 \xrightarrow{l} q'_1\right) \end{cases}$$

The relations $\sim_k^{col}$ (for $k \in \mathbb{N}$) are obviously equivalence relations on Q. The following lemma states well-known facts that are useful for the design of the

bisimulation-based extrapolation. Knuth's up-arrow notation is used in item (*iii*) to denote iterated exponentials: for any $a, k \in \mathbb{N}$, $(a \uparrow\uparrow k)$ is the function $f^k = \underbrace{f \circ \cdots \circ f}_{k \text{ times}}$ where $f : \mathbb{N} \to \mathbb{N}$ is the function defined by $f(x) = a^x$.

Lemma B.6. *For any finite automaton $\mathcal{D} = \langle Q, \Sigma, \to, I, F\rangle$ and equivalence relation col on Q, the three following assertions hold:*

(i) we have $\sim_k^{col} = \sim_{|Q|}^{col}$ for every $k \geq |Q|$.
(ii) if col satisfies $(q, q') \in col \Rightarrow (q \in F \Leftrightarrow q' \in F)$ then $L(\mathcal{D}/\sim_{|Q|}^{col}) = L(\mathcal{D})$.
(iii) for every $k \in \mathbb{N}$, we have $|Q/\sim_k^{col}| \leq (2^{|\Sigma|+1} \uparrow\uparrow k)(|Q/col|)$.

Proof. Let us first prove (*i*). We derive from Definition B.5 that $\sim_k^{col} \supseteq \sim_{k+1}^{col}$, hence, $|Q/\sim_k^{col}| \leq |Q/\sim_{k+1}^{col}|$ for every $k \in \mathbb{N}$. Since $1 \leq |Q/\sim_k^{col}| \leq |Q|$ for all $k \in \mathbb{N}$, we obtain that there exists $k \leq |Q|$ such that $|Q/\sim_k^{col}| = |Q/\sim_{k+1}^{col}|$. We arrive at $\sim_k^{col} \not\supset \sim_{k+1}^{col}$ and hence $\sim_k^{col} = \sim_{k+1}^{col}$. We deduce from Definition B.5 that $\sim_k^{col} = \sim_{k'}^{col}$ for all $k' \geq k$, which concludes the proof of (*i*) as $k \leq |Q|$.

Let us now prove assertion (*ii*). Assume that for each $(q, q') \in col$, it holds that $q \in F$ if and only if $q' \in F$. Let us shortly write $\sim$ in place of $\sim_{|Q|}^{col}$. Since $L(\mathcal{D}) \subseteq L(\mathcal{D}/\sim)$, we only have to show that $L(\mathcal{D}/\sim) \subseteq L(\mathcal{D})$. Consider any word $l_0 \cdots l_{h-1}$ accepted by $\mathcal{D}/\sim$. There exists $q_0, \ldots, q_h \in Q$ such that $[\,q_0\,]_\sim \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} [\,q_h\,]_\sim$ is a path in $\mathcal{D}/\sim$ and such that $q_0 \in I$ and $q_h \in F$. Remark that for any $q, q' \in Q$ and any $l \in \Sigma$, if $[\,q\,]_\sim \xrightarrow{l} [\,q'\,]_\sim$ is a transition in $\mathcal{D}/\sim$ then there exists $q'' \in [\,q'\,]_\sim$ such that $q \xrightarrow{l} q''$ is a transition in $\mathcal{D}$. Indeed, if $[\,q\,]_\sim \xrightarrow{l} [\,q'\,]_\sim$ then we have $r \xrightarrow{l} r'$ for some $r \in [\,q\,]_\sim$ and $r' \in [\,q'\,]_\sim$. Since $\sim = \sim_{|Q|}^{col} = \sim_{|Q|+1}^{col}$, we get from Definition B.5 that there exists $q'' \in [\,q'\,]_\sim$ such that $q \xrightarrow{l} q''$. An immediate induction along the path π shows that there exists $q'_0 \in [\,q_0\,]_\sim, \ldots, q'_h \in [\,q_h\,]_\sim$ such that $q'_0 \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} q'_h$ is a path in $\mathcal{D}$ with $q'_0 = q_0$. Since $q_h \sim q'_h$, we get that $(q_h, q'_h) \in col$ and hence $q'_h \in F$. We deduce that $l_0 \cdots l_{h-1}$ is accepted by $\mathcal{D}$. We have thus shown that $L(\mathcal{D}/\sim) \subseteq L(\mathcal{D})$.

To prove (*iii*), we first introduce some additional notations. Define the function $pre : \Sigma \times \wp(Q) \to \wp(Q)$ by $pre(l, U) = \left\{ q \in Q \;\middle|\; \exists u \in U : q \xrightarrow{l} u \right\}$. For any $\mathcal{U} \subseteq \wp(Q)$, the equivalence relation $\sim_{\mathcal{U}}$ on Q "generated" by $\mathcal{U}$ is defined by: $q_1 \sim_{\mathcal{U}} q_2$ if for every $U \in \mathcal{U}$, we have $q_1 \in U$ if and only if $q_2 \in U$. It follows from Definition B.5 that, for every $k \in \mathbb{N}$, the following equality holds:

$$\sim_{k+1}^{col} \;=\; \sim_k^{col} \cap \sim_{\mathcal{U}_k} \qquad \text{where } \mathcal{U}_k = \left\{ pre(l, U) \;\middle|\; l \in \Sigma, U \in Q/\sim_k^{col} \right\}$$

Let us write $s_k = |Q/\sim_k^{col}|$ for all $k \in \mathbb{N}$. We deduce from the above equality that $s_{k+1} \leq s_k \cdot |Q/\sim_{\mathcal{U}_k}|$ for every $k \in \mathbb{N}$. Since $|\mathcal{U}_k| \leq \Sigma \cdot s_k$, we get that $\sim_{\mathcal{U}_k}$ has at most $2^{|\Sigma| \cdot s_k}$ equivalence classes, and we derive that $s_{k+1} \leq s_k \cdot 2^{|\Sigma| \cdot s_k}$. We obtain that $s_{k+1} \leq 2^{(|\Sigma|+1) \cdot s_k} = \left(2^{|\Sigma|+1}\right)^{s_k}$ for every $k \in \mathbb{N}$. As $s_0 = |Q/col|$, we arrive at $s_k \leq (2^{|\Sigma|+1} \uparrow\uparrow k)(|Q/col|)$ for all $k \in \mathbb{N}$. □

Remark that for every $k \in \mathbb{N}$, the relation $\sim_k^{col}$ is an equivalence relation on Q that is contained in *col*. It follows from Proposition B.3 that for any n-dim queue decision diagram $\mathcal{D}$ for M and for any equivalence relation *col* on Q with $col \subseteq \approx_{\mathcal{D}}$, the quotient $\mathcal{D}/\sim_k^{col}$ is an n-dim queue decision diagram for M.

The last ingredient to obtain an extrapolation is the choice of an adequate equivalence relation *col*. Items (i) and (ii) of Lemma B.6 suggest that *col* should satisfy $(q, q') \in col \Rightarrow (q \in F \Leftrightarrow q' \in F)$. We therefore consider the coloring defined as follows, which will be the standard equivalence relation *col* in our discussion. Given an n-dim queue decision diagram $\mathcal{D} = \langle Q, M \cup \{\in\!\!\!\!\!\!\in\}, \to, I, F \rangle$ for M, we define for every $1 \leq i \leq n$ the sets I_i and F_i by:

$$I_1 = I \qquad I_i = \left\{ q \in Q_i \;\middle|\; \exists q' \in Q : q' \xrightarrow{\in\!\!\!\!\!\!\in} q \right\} \qquad (\text{for } i > 1)$$

$$F_n = F \qquad F_i = \left\{ q \in Q_i \;\middle|\; \exists q' \in Q : q \xrightarrow{\in\!\!\!\!\!\!\in} q' \right\} \qquad (\text{for } i < n)$$

Intuitively, the sets I_i and F_i are respectively the sets of initial and final states for the queue i. The *standard coloring* for $\mathcal{D}$ is the equivalence relation *std* "generated" by the sets Q_i and F_i, formally defined by:

$$(q, q') \in std \quad \text{if} \quad \forall i \in \{1, \ldots, n\} : (q \in Q_i \Leftrightarrow q' \in Q_i) \wedge (q \in F_i \Leftrightarrow q' \in F_i)$$

Our definition of standard coloring is a variant of the one in [LGJJ06], where it was defined as the equivalence relation "generated" by the sets Q_i, I_i and F_i. Note that the quotient $\mathcal{D}/\sim_k^{std}$ is in $\mathcal{Q}dd\,(M, n)$ for every $k \in \mathbb{N}$, since $std \subseteq \approx_{\mathcal{D}}$. We arrive at the following definition.

Definition B.7. *The* bisimulation extrapolation *is the function ρ from $\mathbb{N}$ to the function set $\mathcal{Q}dd\,(M, n) \to \mathcal{Q}dd\,(M, n)$ defined by $\rho_k(\mathcal{D}) = \mathcal{D}/\sim_k^{std}$.*

Proposition B.8. *The function ρ is a restricted* QDD*-extrapolation.*

Proof. Let $\mathcal{D} = \langle Q, M \cup \{\in\!\!\!\!\!\!\in\}, \to, I, F \rangle$ be an n-dim queue decision diagram for M. For every $k \in \mathbb{N}$, the relation $\sim_k^{std}$ is an equivalence relation on Q. Therefore, the quotient $\rho_k(\mathcal{D}) = \mathcal{D}/\sim_k^{std}$ satisfies $L(\mathcal{D}) \subseteq L(\rho_k(\mathcal{D}))$, which proves condition (i) of Definition B.1. Observe that the standard coloring *std* satisfies $(q, q') \in std \Rightarrow (q \in F \Leftrightarrow q' \in F)$. According to Lemma B.6, it holds that $L(\rho_k(\mathcal{D})) = L(\mathcal{D})$ for all $k \geq |Q|$, which proves condition (ii) of Definition B.1. We have thus shown that ρ is a QDD-extrapolation.

Notice that the standard coloring *std* satisfies $|Q/std| \leq 2n$ for every n-dim queue decision diagram $\langle Q, M \cup \{\in\!\!\!\!\!\!\in\}, \to, I, F \rangle$ for M. For any bound $b \in \mathbb{N}$, the set of all finite automata $\mathcal{D} = \langle Q, \Sigma, \to, I, F \rangle$ with $|Q| \leq b$ and $\Sigma = M \cup \{\in\!\!\!\!\!\!\in\}$ is finite up to automata isomorphism. We deduce from item (iii) of Lemma B.6 that ρ is restricted. □

Remark B.9. The proof of the previous proposition only relies on the two following properties of the standard coloring: $|Q/std|$ is uniformly bounded and $(q, q') \in std \Rightarrow (q \in F \Leftrightarrow q' \in F)$. Therefore, any equivalence relation contained in $\approx_{\mathcal{D}}$ and satisfying these two properties may be used in place of *std* (for instance, the standard coloring of [LGJJ06]).

Remark B.10. We mentioned in Remark 4.4 page 13 that extrapolations are closed under round-robin combination. Consider in particular the functions ρ' and ρ'' from $\mathbb{N}$ to the function set $\mathcal{Q}dd(M,n) \to \mathcal{Q}dd(M,n)$ defined by $\rho'_0(\mathcal{D}) = \rho''_0(\mathcal{D}) = \mathcal{D}/{\approx_{\mathcal{D}}}$, $\rho'_k = \rho_{k-1}$ and $\rho''_k = \rho_k$ for all $k \geq 1$. The functions ρ' and ρ'' are also restricted QDD-extrapolations. Compared to the extrapolation ρ of Definition B.7, the extrapolations ρ' and ρ'' provide the coarsest quotient-based QDD approximation when the parameter k is zero.

B.2 Bounded Languages-based Extrapolation

We present in this subsection the extrapolation underlying the automata abstraction function based on finite-length languages introduced in [BHV04] for minimal deterministic automata. We extend in this subsection the automata abstraction function based on finite-length languages of [BHV04] to arbitrary finite automata and to QDDs.

We first introduce some additional notations. Consider a finite automaton $\mathcal{D} = \langle Q, \Sigma, \to, I, F\rangle$. We write $|w|$ for the *length* of any word $w \in \Sigma^*$. Given a bound $b \in \mathbb{N}$, the *accepted language* of $\mathcal{D}$ *up to* b is the set $L^{\leq b}(\mathcal{D})$ of all words $w \in L(\mathcal{D})$ of length at most b, formally $L^{\leq b}(\mathcal{D}) = \{w \in L(\mathcal{D}) \mid |w| \leq b\}$. For any state $q \in Q$ and for any subset $S \subseteq Q$, we denote by $\mathcal{D}[q,S]$ the finite automaton $\mathcal{D}[q,S] = \langle Q, \Sigma, \to, \{q\}, S\rangle$, and we shortly write $L^{\leq b}(\mathcal{D}, q, S)$ in place of $L^{\leq b}(\mathcal{D}[q,S])$.

Definition B.11. *Let $\mathcal{D} = \langle Q, \Sigma, \to, I, F\rangle$ be a finite automaton, the* language equivalence of depth k *is the relation $\sim_k$ on Q defined by:*

$$q_1 \sim_k q_2 \qquad \textit{if} \qquad L^{\leq k}(\mathcal{D}, q_1, F) \;=\; L^{\leq k}(\mathcal{D}, q_2, F)$$

The relation $\sim_k$ (for $k \in \mathbb{N}$) is obviously an equivalence relation on Q. Notice that $\sim_{k+1} \subseteq \sim_k$ for every $k \in \mathbb{N}$. The following two lemmata state well-known properties that are useful for the design of the extrapolation based on bounded languages.

Lemma B.12. *Consider a finite automaton $\mathcal{D} = \langle Q, \Sigma, \to, I, F\rangle$. For every $k \in \mathbb{N}$ and $q \in Q$, it holds that:*

$$L^{\leq k}(\mathcal{D}, q, F) \quad = \quad L^{\leq k}\Big(\mathcal{D}/{\sim_k}, [\,q\,]_{\sim_k}, \{[\,q_f\,]_{\sim_k} \mid q_f \in F\Big)$$

Proof. We proceed by mutual inclusion. If $Q' \subseteq Q$, we denote by $\widetilde{Q'}_k$ the set $\{[\,q'\,]_{\sim_k} \mid q' \in Q'\}$. The inclusion $L^{\leq k}(\mathcal{D}, q, F) \subseteq L^{\leq k}(\mathcal{D}/{\sim_k}, [\,q\,]_{\sim_k}, \widetilde{F}_k)$ follows from the fact that $[\,q_0\,]_{\sim_k} \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} [\,q_h\,]_{\sim_k}$ is a path in $\mathcal{D}/{\sim_k}$ for any path $q_0 \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} q_h$ in $\mathcal{D}$. To prove the reverse inclusion, we show by induction on k that $L^{\leq k}(\mathcal{D}, q, F) \supseteq L^{\leq k}(\mathcal{D}/{\sim_k}, [\,q\,]_{\sim_k}, \widetilde{F}_k)$ for every $q \in Q$.

We first prove the basis; let $k = 0$ and let $q \in Q$. By definition $q \in F$ iff $[\,q\,]_{\sim_0} \in \widetilde{F}_k$ iff $L^{\leq 0}(\mathcal{D}, q, F) = \{\varepsilon\} = L^{\leq 0}(\mathcal{D}/{\sim_0}, [\,q\,]_{\sim_0}, \widetilde{F}_0)$.

We now prove the induction step. Consider any $k \in \mathbb{N}$ and assume that $L^{\leq k}(\mathcal{D}, q, F) \supseteq L^{\leq k}(\mathcal{D}/{\sim_k}, [\,q\,]_{\sim_k}, \widetilde{F}_k))$ for every $q \in Q$. Let $q \in Q$ and let $w \in L^{\leq k+1}(\mathcal{D}/{\sim_{k+1}}, [\,q\,]_{\sim_{k+1}}, \widetilde{F}_{k+1}))$ of length h. There exists $q_0, \ldots, q_h \in Q$ such that $[\,q_0\,]_{\sim_{k+1}} \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} [\,q_h\,]_{\sim_{k+1}}$ is a path in $\mathcal{D}/{\sim_{k+1}}$ and such that $w = l_0 \cdots l_{h-1}$, $q_0 \sim_{k+1} q$ and $q_h \in \widetilde{F}$. Since $\sim_{k+1} \subseteq \sim_k$, we obtain that $[\,q_0\,]_{\sim_k} \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} [\,q_h\,]_{\sim_k}$ is a path in $\mathcal{D}/{\sim_k}$. Therefore, if $|w| \leq k$ then $w \in L^{\leq k}(\mathcal{D}/{\sim_k}, [\,q\,]_{\sim_k}, \widetilde{F}_k))$, and we deduce from the induction hypothesis that $w \in L^{\leq k}(\mathcal{D}, q, F)$, hence, $w \in L^{\leq k+1}(\mathcal{D}, q, F)$.

Let us now assume that $|w| = h = k + 1$. Since $[\,q_0\,]_{\sim_{k+1}} \xrightarrow{l_0} [\,q_1\,]_{\sim_{k+1}}$, there exists $q'_0 \in [\,q_0\,]_{\sim_{k+1}}$ and $q'_1 \in [\,q_1\,]_{\sim_{k+1}}$ such that $q'_0 \xrightarrow{l_0} q'_1$ is a transition in $\mathcal{D}$. Let $w' = l_1 \cdots l_k$. Recall that $[\,q_1\,]_{\sim_k} \xrightarrow{l_1} \cdots \xrightarrow{l_k} [\,q_{k+1}\,]_{\sim_k}$ is a path in $\mathcal{D}/{\sim_k}$ with $q_1 \sim_k q'_1$ and $q_{k+1} \in F$. Therefore, we have $w' \in L^{\leq k}(\mathcal{D}/{\sim_k}, [\,q'_1\,]_{\sim_k}, \widetilde{F}_k))$ and it follows from the induction hypothesis that $w' \in L^{\leq k}(\mathcal{D}, q'_1, F)$. We deduce that $w \in L^{\leq k+1}(\mathcal{D}, q'_0, F)$. Moreover, $L^{\leq k+1}(\mathcal{D}, q, F) = L^{\leq k+1}(\mathcal{D}, q'_0, F)$ as $q \sim_{k+1} q_0 \sim_{k+1} q'_0$. We conclude that $w \in L^{\leq k+1}(\mathcal{D}, q, F)$. □

Lemma B.13. *For any finite automaton $\mathcal{D} = \langle Q, \Sigma, \to, I, F\rangle$, there exists $K \in \mathbb{N}$ such that the two following assertions hold:*

(i) we have $\sim_k = \sim_K$ for every $k \geq K$.
(ii) $L(\mathcal{D}/{\sim_K}) = L(\mathcal{D})$.

Proof. It follows from Definition B.5 that $\sim_k \supseteq \sim_{k+1}$ for every $k \in \mathbb{N}$. Since there are only finitely many equivalence relations on Q, we get that there is K in $\mathbb{N}$ such that $\sim_k = \sim_K$ for every $k \geq K$.

Since $L(\mathcal{D}) \subseteq L(\mathcal{D}/{\sim})$, we only have to show that $L(\mathcal{D}/{\sim}) \subseteq L(\mathcal{D})$. Consider any word w accepted by $\mathcal{D}/{\sim}$. Let $k = \max(|w|, K)$. There exists $q_i \in I$ and $q_f \in F$ such that $w \in L^{\leq k}(\mathcal{D}/{\sim}, [\,q_i\,]_{\sim}, \{[\,q_f\,]_{\sim}\})$. Let $\widetilde{q_f} = [\,q_f\,]_0$ and remark that $\widetilde{q_f} \subseteq F$. Note that $w \in L^{\leq k}(\mathcal{D}/{\sim}, [\,q_i\,]_{\sim}, \widetilde{F}_k)$. Since $k \geq K$, it holds that $\sim_k = \sim$ and we deduce from Lemma B.12 that $w \in L^{\leq k}(\mathcal{D}, q_i, \widetilde{q_f})$. As $q_i \in I$ and $\widetilde{q_f} \subseteq F$, we come to $w \in L(\mathcal{D})$. □

When defining this extrapolation for a n-dim queue decision diagram $\mathcal{D} = \langle Q = Q_1 \uplus Q_2 \uplus \cdots \uplus Q_n, \Sigma, \to, I, F\rangle$, we ensure that states of the different parts of the automaton do not merge. Moreover, if the size of $\mathcal{D}$ is far greater than k, The relation $\sim_k$ does not distinguish states that belongs to the "beginning" of $\mathcal{D}$, i.e. states of Q_1, Q_2, etc. That is why we consider not only the *global* language equivalence of depth k, but also n *local* language equivalence of depth k, one for each queue number. The global language equivalence is the relation $\sim_k$ defined like before. The local language equivalence $\sim_{i,k}$ has the same definition when the automaton is restricted to Q_i, in other words we consider, for any state $q \in Q_i$, the language $L^{\leq k}(\mathcal{D}_i, q_i, F_i)$ where $\mathcal{D}_i = \langle Q_i, \Sigma, \to_i, I_i, F_i\rangle$, $\to_i$ being the restriction of $\to$ to $Q_i \times \Sigma \times Q_i$.

We thus define the relation $\sim_k^{Qdd}$ on states of $\mathcal{D}$ as:

$$q_1 \sim_k^{Qdd} q_2 \quad \text{if} \quad \exists i \in [1,n], \quad \begin{cases} q_1 \in Q_i \ \wedge \ q_2 \in Q_i \\ \wedge L^{\leq k}(\mathcal{D}, q_1, F) \ = \ L^{\leq k}(\mathcal{D}, q_2, F) \\ \wedge L^{\leq k}(\mathcal{D}_i, q_1, F_i) \ = \ L^{\leq k}(\mathcal{D}_i, q_2, F_i) \end{cases}$$

Remark that for every $k \in \mathbb{N}$, the relation $\sim_k^{Qdd}$ is an equivalence relation on Q, and that the lemmata demonstrated for $\sim_k$ also hold for $\sim_k^{\mathcal{D}}$. It follows from Proposition B.3 that for any n-dim queue decision diagram $\mathcal{D}$ for M, the quotient $\mathcal{D}/\sim_k^{Qdd}$ is an n-dim queue decision diagram for M.

Definition B.14. *The* bounded language extrapolation *is the function ξ from $\mathbb{N}$ to the function set $\mathcal{Q}dd\,(M,n) \to \mathcal{Q}dd\,(M,n)$ defined by $\xi_k(\mathcal{D}) = \mathcal{D}/\sim_k^{Qdd}$.*

Proposition B.15. *The function ξ is a restricted* QDD*-extrapolation.*

Proof. Let $\mathcal{D} = \langle Q, M \cup \{\in\}, \to, I, F\rangle$ be an n-dim queue decision diagram for M. For every $k \in \mathbb{N}$, the relation $\sim_k^{Qdd}$ is an equivalence relation on Q. Therefore, the quotient $\xi_k(\mathcal{D}) = \mathcal{D}/\sim_k^{Qdd}$ satisfies $L(\mathcal{D}) \subseteq L(\xi_k(\mathcal{D}))$, which proves condition (i) of Definition B.1. According to Lemma B.13, there exists $K \in \mathbb{N}$ such that $L(\xi_k(\mathcal{D})) = L(\mathcal{D})$ for all $k \geq K$, which proves condition (ii) of Definition B.1. We have thus shown that ξ is a QDD-extrapolation.

It is easily seen that for any $k \in \mathbb{N}$, the equivalence relation $\sim_k$ has at most 2^{Σ^k} equivalence classes, because Σ^k is the maximal number of distinct words of length up to k. By definition, $\sim_k^{Qdd}$ has at most $n \times (2^{\Sigma^k})^{n+1}$ equivalence classes. For any bound $b \in \mathbb{N}$, the set of all QDD $\mathcal{D} = \langle Q, \Sigma, \to, I, F\rangle$ with $|Q| \leq b$ and $\Sigma = M \cup \{\in\}$ is finite up to automata isomorphism. We conclude that ξ is restricted. □

B.3 Comparison between the two Extrapolations

We consider an n-dimension QDD given by $\mathcal{D} = \langle Q, \Sigma, \to, I, F\rangle$ with $Q = \cup_{i=0}^n Q_i$ and $F = \cup_{i=0}^n F_i$. In this context, the standard coloring is $std = \cup_{i=0}^n (F_i \times F_i) \cup \cup_{i=0}^n ((Q_i \setminus F_i) \times (Q_i \setminus F_i))$. Let us denote by B_k the bisimulation equivalence of depth k defined in Definition B.5, with coloring *std*. Let us denote by T_k the language equivalence of depth k defined in Definition B.11.

Lemma B.16. *For all $k \in \mathbb{N}$, $B_k \subseteq T_k$, i.e., forall $q, q' \in Q$ it holds that $(q,q') \in B_k \Rightarrow (q,q') \in T_k$.*

Proof. By induction on k. The basis is trivial, since $B_0 = T_0 = std$. Thus, let $k \in \mathbb{N}$ and assume that $B_k \subseteq T_k$.

Conversely, take a pair $(q_1, q_2) \in B_{k+1}$. Since $B_{k+1} \subseteq B_k \subseteq T_k$ we have $L^{\leq k}(\mathcal{D}, q_1, F) = L^{\leq k}(\mathcal{D}, q_1, F)$. Now consider a word $w \in L^{\leq k+1}(\mathcal{D}, q_1, F)$, and let us write $w = l \cdot w'$. There exists $q_1' \in Q$ such that $q_1 \xrightarrow{l} q_1'$ and $w' \in L^{\leq k}(\mathcal{D}, q_1', F)$. Moreover, as $(q_1, q_2) \in B_{k+1}$, there exists q_2' such that $q_2 \xrightarrow{l} q_2'$

and $(q_1', q_2') \in B_k$. Therefore, $(q_1', q_2') \in T_k$, and we get that $w' \in L^{\leq k}(\mathcal{D}, q_2', F)$, hence, $w \in L^{\leq k+1}(\mathcal{D}, q_2, F)$. It follows that $L^{\leq k+1}(\mathcal{D}, q_1, F) \subseteq L^{\leq k+1}(\mathcal{D}, q_2, F)$. We obtain by symmetry of B_{k+1} that $L^{\leq k+1}(\mathcal{D}, q_2, F) = L^{\leq k}(\mathcal{D}, q_1, F)$. □

Note that this proposition does not allow to deduce that B_k is a "better" equivalence relation than T_k with respect to the CEGAR algorithm. In fact, if the quotient of $\mathcal{D}$ by T_k merge more states, it may give a more general invariant and thus rule out more spurious examples.

C Example Protocols

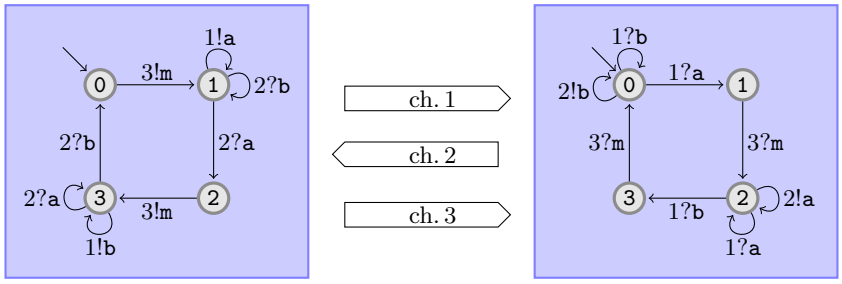
We present in this section the suite of protocols (except for the c/d protocol which was already introduced in Example 2.1) on which we tested our tool. Each protocol is specified as a system of communicating processes. In each case, the resulting fifo system is the asynchronous product of the processes. The queues are initially empty, and each process has a single initial state that is graphically indicated by an arrow with no source state. We provide with each protocol the set of bad configurations used in our experimental evaluation.

Alternating Bit Protocol
Figure 11a presents the classical example protocol for automatic verification of finite fifo systems, in the formalization of [LGJJ06]. The two participating peers exchange control data over the channels 1 and 2 as well as data over channel 3. We checked that the sender and the receiver (left hand-side and right hand-side of Figure 11a, espectively) are loosely synchronized. Formally, the safety property is given by the following set of good control states, which should be the only reachable ones: $\{\mathtt{00}, \mathtt{10}, \mathtt{11}, \mathtt{12}, \mathtt{22}, \mathtt{32}, \mathtt{33}, \mathtt{30}\}$.

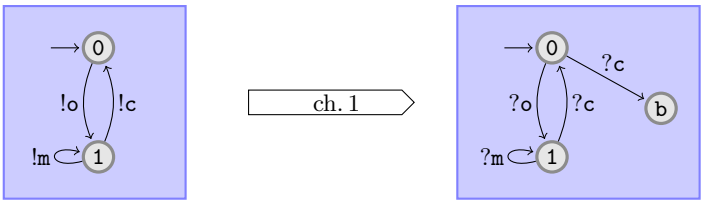
Nested Connection/Disconnection Protocol
Systems with nested loops overburden standard acceleration techniques, which rely on the analysis of simple loops and cannot accelerate nested loops. We have extended the connection/disconnection protocol with simple loops to exchange data (message `m`) from the client to the server, see Figure 11b. This variant does not have the disconnect transitions, as otherwise the example would be unsafe and, hence, easier to verify with a CEGAR approach. We checked the same safety property as the c/d protocol, directly specified here by the state (`b`)`ad` of the server, which should not be reachable.

Non-Regular Protocol
Figure 11c presents a simple example where the reachability set is not recognizable. Indeed, the set of reachable queue contents in control state `00` is $\{(a^m, \varepsilon, b^m, \varepsilon) \mid m \in \mathbb{N}\}$ which is well known to be not recognizable. The safety property is given explicitly by the control state `02`, which should not be reachable.

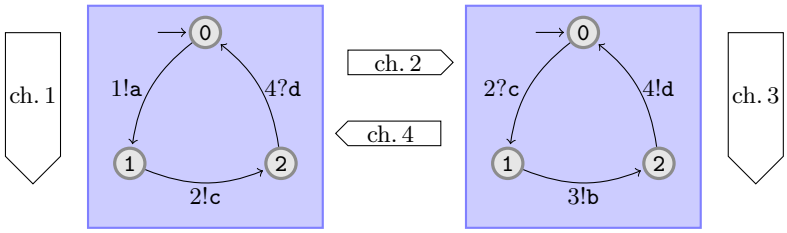
Remark C.1. Even though we only compute invariants from recognizable subsets, our approach is able to verify the safety property on this non-regular example. Other techniques that are based on recognizable subsets, but that rely on an exact computation of the reachability set (e.g., symbolic exhaustive exploration with QDDs and acceleration [BG99]) are not able to handle finite fifo systems with non-regular configurations. On the other hand, our technique is limited to safety properties that can be proven with recognizable invariants.



(a) Alternating Bit Protocol



(b) Extension of the C/D Protocol with Nested Loops for Data

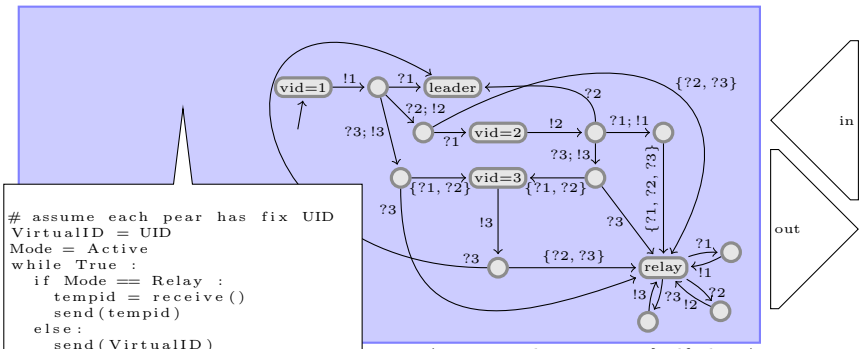


(c) Non-Regular Protocol using Channels 1 & 3 like Pushdown Stacks

Fig. 11: Example Protocols

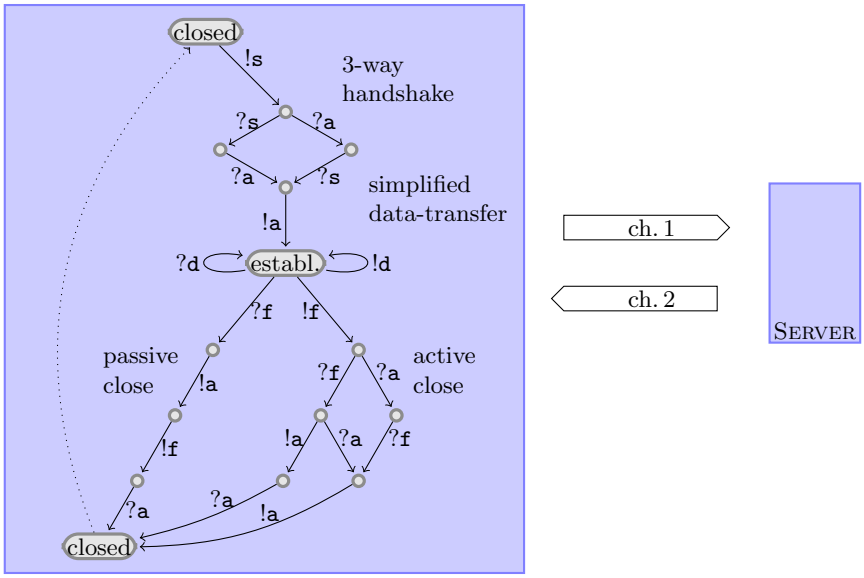
Peterson's Leader Election
This is a translation of Peterson's leader election algorithm [Pet82] (viz. Figure 12a that includes pseudocode taken from [Ans08]) into a finite fifo systems. The algorithm is modeled as a set of finite state automata which are executed distributively (and asynchronously) in a ring topology. We check whether more than one process asserts that he is the leader. We fix for our benchmark the number of peers to 3 as our approach does not permit parametrized model checking.

Simplified TCP
Based on the underlying state transition of the TCP protocol and by ignoring all the additional timing constraints as well as the sophisticated data transport




```
# assume each pear has fix UID
VirtualID = UID
Mode = Active
while True :
  if Mode == Relay :
    tempid = receive()
    send(tempid)
  else:
    send(VirtualID)
    uid2  = receive()
    if VirtualID == id2 :
      announce("leader!")
    else:
      send(id2);
      uid3 = receive();
      if id2>max{VirtualID,id3}:
          VirtualID = id2;
      else:
          Mode = Relay;
```

(a) Leader Election in an Asynchronous Ring following Peterson (Peer 1)



(b) Simplified Transmission Control Protocol

Fig. 12: More Example Protocols

(sliding windows etc.), we modeled the three-way handshake of TCP as well as the passive/active close in a simple client/server setting with one bidirectional channel.

The diagram in Figure 12b presents only the client in detail, the server is identical except for exchanging send and receive in the 3-way handshake phase. We further restrict ourselves to the messages (`s`)`yn`, (`a`)`ck`, (`f`)`in`, (`d`)`ata` without any additional sequence numbering nor user data.

We verified that the connection establishment and termination work by checking whether one of the peers remains in the *closed* state whereas the other assumes the connection to be *established*.

Server with Two Clients
This is a simple extension of the (simplified) TCP protocol, where we verify the correctness of connection establishment and termination in the case of a second client that uses the same channels as the original client, but with distinct messages.

Token Ring Protocol
Figure 13 is an example of a token passing protocol of n identical processes, set in a ring architecture. At the beginning, each process has 0 or 1 token (local states `0` or `1`). A process is in a "bad" configuration when it has two tokens (local state `b`). Therefore, it sends an alert message a before sending a token t. When a process receives an alert message, it ignores it (if it has no token) or sends immediately its token to the following process, without an alert message. Thus the only outgoing transition of local state `3` is to send a token. For our benchmark, we fix the number of processes to 4.

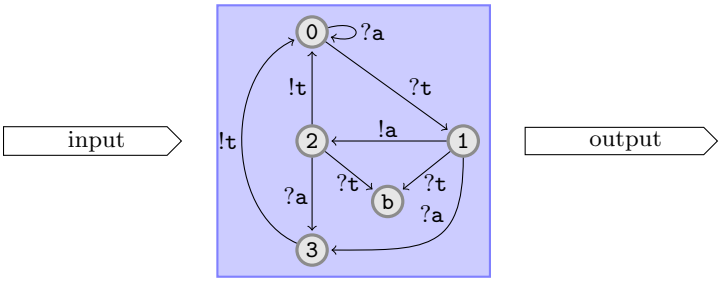


Fig. 13: Generic Peer of Token Ring Protocol (inital state either 0 or 1)

Sliding Window Protocol
The family of sliding window protocols defines a safe way to transfer data over a channel. The size of the sliding window is a priori fixed (as in our case), or adaptively changes dynamically. At each moment, the sender restricts the

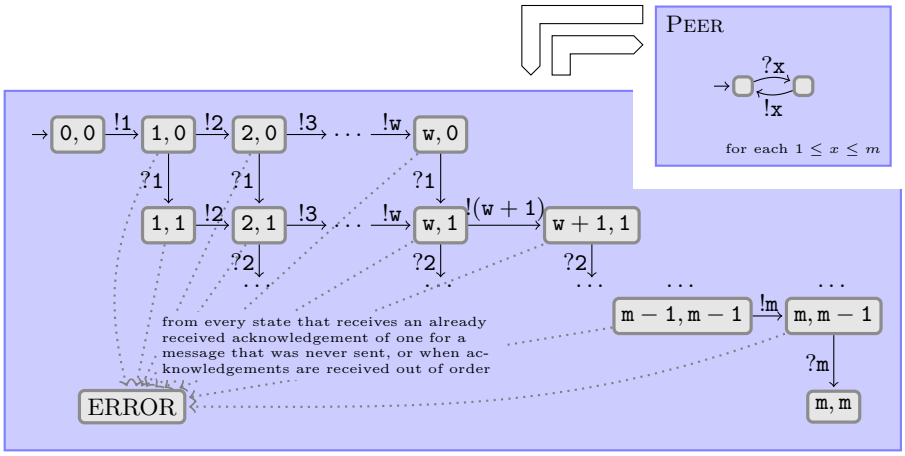


Fig. 14: Simple Sliding Window Protocol (m message length, w fix window)

number of unacknowledged messages in the queue towards the receiver to be smaller than the window size.

The instantiation of the sliding window protocol in the following benchmarks has a fixed message length of 10 and a window size of 2. If the sender receives an acknowledgment for a packet that was not already sent or that was already acknowledged, our protocol aborts the transfer by entering an error state. Further, receiving acknowledgments not in the order of sent packages also leads to the error state, whose reachability will be checked.

A Bounded Retransmission Protocol

The *bounded retransmission protocol* (Brp) was designed by Philips to transmit (large) data packages over an unreliable medium by splitting packages into frames and each frame into a sequence of small chunks. It is based on the Abp and allows for a bounded number of retransmissions of each frame. Due to both its simplicity and its practical relevance, it can also be seen as one of the standard examples in protocol verification , e.g., a simple Brp model is also included as standard example with Trex [TReX].

As our framework does not allow timers, we combine ideas from the previous mentioned models: we focus only the alternating bit part of the exchange of frames, hence, ignore the transmission of data; the expiration of timers is modeled as non-deterministic choice and the intrinsic synchronization of the two processes timers are explicitly modeled by a synchronization via a different pair of channels, that assures that both timers are run out and both processes agree that they restart the transmission of a frame by additionally emptying the channels. See Figure 15 for details, where `f` is the (f)irst message sent and `l` the (l)ast and the (s)ync message `s` is the only message exchanged over the third channel.

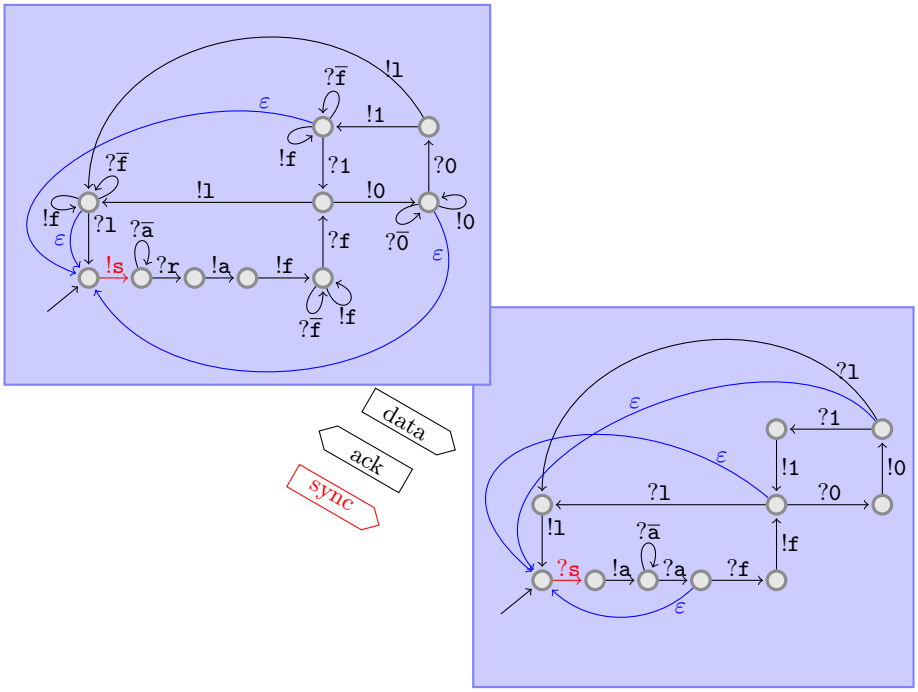


Fig. 15: A Bounded Retransmission Protocol

We verify the safety properties of the underlying ABP (as before), as well as prove that our model of synchronizing the timers via an extra channel and forcing the emptiness of the channels before sending the `first` message is accurate.

A Variant of POP3

This Protocol is based on RFC 1939. Here, we abstract the exchanged data even further and only transmit commands that either result in an error message or an acknowledgment (maybe followed by a data token). We only focus one singular connection, and verify that it is impossible that the client is in the "transmission" phase (i.e., already passed the username/password check) and the server still is in "authentication" phase, or vice versa.